CSC 411, Fall 2006 — Assignment #2

*Due at **start** of tutorial time on November 3. Note that this assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. Handing in work that is not your own is a serious academic offense. Fabricating results, such handing in fake output that was not actually produced by your program, is also an academic offense.*

For this assignment, you will train a multilayer perceptron neural network to fit data from a project I am working on with Lev Tarasov and Richard Peltier in the physics department.

The data concerns the output of a complex simulation of glaciation in North America over the last 250,000 years. The aim of the project is to reconstruct the history of glaciation over that time period. Many of the parameters of the simulator are unknown, however. These parameters control the dynamics of ice movement, aspects of the assumed climate over this time period, etc. These parameters need to be estimated on the basis of how well the simulated history of glaciation using given parameter values matches the available data on past glaciation. Primarily, the data concerns the ages of fossilized mussels found at various elevations above present sea level, from which one can infer something about how much ice was present, pushing the land down and leading to an apparent rise in sea level.

For this assignment, however, we will look at another constraint on the simulator parameters, which is that they should not produce ice sheets whose thickness is less than is plausible. In particular, for given values of the 23 simulator parameters, the simulator may or may not produce an ice sheet 25,000 years ago that is at least 200m thick over Hudson's Bay. If it doesn't, we want to regard those parameters as not being possible values.

Neural networks come into the project because the simulator takes quite a long time to run — several hours on a vector supercomputer. Because of this, it's not feasible to explore possible parameters values by just running the simulator for each set of parameter values that are considered. Instead, we use past runs of the simulator to train neural networks that predict what the simulator would do if run with given parameters. The predictions aren't going to be perfect, of course, but the hope is that they will be good enough to choose some good sets of parameters, which can then be checked using a relatively small number of actual simulator runs.

We therefore need a neural network model that will predict whether or not the Hudson's Bay constraint will be violated, given the values of the 23 simulator parameters as inputs.

For purposes of this assignment, I have divided the set of past runs that are available into a training set of 1708 cases and a test set of 2562 cases. You should try training on the training set, and then (only at the very end) see how well the model does on the test set. You should evaluate predictive performance primarily by the classification error rate.

The data is available from the web page, and in the following files in /u/radford/course/csc411/ass2 on the CDF computer system:

a2trnx	The 23 simulator parameters for training cases
a2tstx	The 23 simulator parameters for test cases
a2trny	The Hudson's Bay indicator for training cases
a2tsty	The Hudson's Bay indicator for test cases

Inputs for training cases are stored one per line, with each line of a2trnx or a2tstx containing 23 numbers. Each line of a2trny or a2tsty contains one number (0 or 1).

You should fit an MLP model with a single layer of 15 hidden units. The binary target value should be modeled by $P(y = 1|x) = 1/(1 + \exp(-o))$, where o is the output of the network with inputs x.

You should train this model by simple gradient descent, computing the derivatives of the log likelihood with respect to the network parameters using backpropagation, as described in the lecture slides, using a program you write yourself. You should use cross validation to decide when to stop training, and which of the parameter sets found during training to use for prediction. For this purpose, you will need to randomly divide the training set into an estimation set and a validation set. For this assignment, use 1/4 of the data for validation and 3/4 for estimation. I have already randomly re-ordered the training cases. For consistency between students, you should all use the first 1/4 of the cases as the validation set and the last 3/4 of the cases as the estimation set.

You should somehow decide when to stop training, using the average log probability of targets in the validation set as a guide. You can make this decision automatically (in your program), or manually. Either way, be careful not to stop too early, since sometimes the probability of validation cases goes down for a bit before going up again. Once you've finished training, you should use results on the validation set to pick which iteration to use. (For this, you will have to have saved the parameter values at all iterations.)

You should expect that you will need 10000 or more iterations, which may take ten minutes or more. To avoid really long computation times, you should be sure to use vector operations in R or Matlab. At a minimum, you can compute the values for all training cases simultaneously, which will be much faster than using a loop in R or Matlab.

You will need to set the learning rate, η, to a value that's small enough to produce more or less stable learning, with the log likelihood usually increasing from one iteration to the next. Of course, you don't want a really small value of η either. Values for η of about 10^{-4} may be about right.

So that this assignment goes a bit beyond standard methods, you should try a variation in which you use a different learning rate for the β parameters than for the γ parameters. See if you can get better results this way.

At the end, you should evaluate how well the methods did by making predictions for the test cases, and seeing what the error rate with these predictions is. (You can also look at average log probability or squared error for these predictions.)

You should hand in a paper listing of your functions, written in R, Matlab, Octave, or some other suitable language (if you check with me first). Your program should be written in a readable style, and be reasonably efficient (though there is no need to be fanatical in this regard). You should also hand in some results from your tests, including plots showing the average log probability for estimation cases and for validation cases for each iteration of your training runs. Finally, you should briefly discuss your results.