IT3708 Homework 3:
Feed-Forward Neural Net with Backpropagation Learning

Purpose: Gain hands-on experience with neural networks and the classic backpropagation algorithm.

1 Assignment

Implement a feed-forward neural network classifier with gradient-descent learning (via the backpropagation algorithm) as explained in class. The details of backpropagation are in the compendium chapter backprop.pdf and in chapter 2 of *The Essence of Neural Networks* (Callan, 1999). Read each thoroughly before beginning this assignment!

Your system will act as a classifier in that attribute values will be encoded into values in the input layer, and the (encoded) correct class should come out on the output layer.

The inputs to your system should include:

1. The topology of the network in terms of how many layers to include and how many nodes are in each layer. It can be assumed that every node in layer k sends an arc to every node in layer k+1, but to no other nodes in any other layers.
2. The transfer function - each neuron should use the same function.
3. The data set.
4. The percentage of the data set that should be used for training, with the rest being used for testing.
5. The attribute in the data set that will constitute the class, and hence the output, of the neural network.
6. The learning rate.
7. The number of training epochs.

Your system must accept all of these **as parameters**. You should **not** need to recompile your system in order to change from a 3-layered to a 4-layered network, for example.

It is recommended that, in addition to its normal inputs from layer k-1, any layer-k neuron, X, should have one auxiliary input from a dummy node whose output is always 1. By learning the weight, W, on the arc between this dummy node and X, the network essentially learns the threshold for the transfer function. For example, if W = -5, then a sigmoidal transfer function would need a total of +5 weighted inputs from the k-1 layer in order to fire (since the sigmoid begins to go high when its summed input exceeds 0). These dummy nodes and weights are evident in most of the examples in the lecture-note file neurorep.ppt.

Run your ANN on at least two different data sets from the Irvine repository (www.ics.uci.edu/ mlearn/MLRepository.html). Each data set must contain at least 100 entries. On each data set, run experiments where you test out the effects of different:

1. network topologies - in particular, different numbers of nodes in the hidden layer. Also, check the difference between using one versus two hidden layers. Test AT LEAST 3 different topologies for each data set.

2. learning rates - select at least 3 different values between 0.1 and 1.

3. transfer functions - try, at least, a sigmoidal and a linear output function.

You do not need to test all permutations of these. You might begin with a simple topology and find the learning rate and transfer function that work best for it. Then use that same rate and function when you test other topologies.

Remember to divide each data set into a training set and a test set. Include about 80% of the data points in the training set and the remaining 20 % in the test set.

For each test, produce a graph that shows the training error on the y axis and the epoch number on the x axis. This will illustrate the progression of gradient-descent learning. Then, for each case, show the total error on the **test** data. This indicates the degree to which the network learns a general mapping, as opposed to just learning the training examples.

For each case, do a second run in which you continue training well past the point where the training error levels off at a low value. Now run the test cases again and see if the test error is actually worse when you overtrain.

2 The Irvine Data Sets

Be aware that some data sets in the repository have missing data. This is also specified on the summary page of the repository, which has basic information about each data set. If you use a file with missing data, you will have to take special measures to insure that your system does not crash on these cases. For this assignment, it is acceptable to go through the files and just fill in the missing data with the average values of the given attribute in the data set. For example, if 3 of 100 cases do not have a *weight* value, then just use the average weight of the other 97 cases for the 3 missing values.

Check each dataset by hand (or with a pre-processing routine) to insure that no unreasonable values have been included. There are rumors that a few of the data sets are a bit *contaminated*.

For qualitative values with no natural ordering, such as the *color* attribute of flowers, it is wise to use a sparse (semi-local) coding for the attribute: each attribute value (e.g. each color) has its own input node. For attributes with a natural ordering among values, such as *size*, a single input neuron may be sufficient, where sizes are simply scaled to values between 0 and 1. However, some cases are better solved by *partitioning* quantitative values into one of k bins, where each bin maps to one input neuron. For example, the quantitative lengths of automobiles might be converted into bins for sub-compact, compact, mid-sized and luxury, with each getting its own input neuron. You will have to experiment to find the right representation for each dataset.

3 Deliverables

1. A overview description of your code. Details of backpropagation itself do not need to be addressed unless your algorithm differs significantly from the standard version discussed in class.

2. Documention of the many test runs that you did, including graphs of training error as a function of the epoch, and the total error on the test cases. Also, for each of the data sets, include a description of

how case information is encoded into values in your input layer and how class information is decoded from the values on your output layer. Pictures are usually quite helpful in illustrating encodings and decodings!

3. Any other interesting observations that you have about backpropagation in general or your experiments in particular.