Cosc 420 – Neural Networks Assignment

Introduction

As per the assignment specification I have designed a fully connected feed forward network with a single layer of hidden neurodes. The first thing I considered when designing the network was the overarching structure of my program and how that should be informed by the operation of the network itself.

I started by thinking about how to represent the various elements required for the network to run and I decided on an implicit representation of the weights between layers. By this I mean that as far as the code is concerned there is no explicit connection between units. My reason for doing so is that when doing some research prior to starting the assignment I found example code that used the java library class HashMap to explicitly represent connections in a rather complicated way and that seemed to be rather excessive. So with this implicit representation in mind I began by designing a Neurode class to encapsulate all of the state and behavior associated with a Neurode.

The Neurode class has a series of data fields to store all of the information required for the operation of a generalized delta rule network. The weights array is of a size determined by the number of units in the following layer, and at each index of the array is a double that stores the weight from this neurode to the neurode in the next layer indicated by the index. For example, in the weights array of a given hidden neurode, x, weights[0] gives the weight of the connection between x and the first neurode in the output layer. This simple array representation of the weights forms the basis of the individual algorithms involved in the learning process.

I use two more double array data fields in the Neurode class. The first is the changes array. I realised that I essentially needed two weights arrays as when calculating the weight changes during the backwards pass of the backpropagation algorithm the original weight values from other layers are required. So on the backwards pass I calculate the weight changes and store them in the changes array. The second extra array is used to store the changes from the previous epoch, this is used for applying momentum to weight changes. The bias weight, activation value, and error value are also represented by double data fields.

Most of the behavior of the Neurode class is apparent when looking at the code. All but two of the methods are used for getting and setting the weights, bias, activation, and error of the Neurode. The methods of note are the constructor and updateWeights(). The constructor method accepts an int that is the size of the next layer of units, this is used to initialise the weight array, and two Boolean flags that are used to determine which layer the Neurode is in. The weights array and bias are initialised with random weights in the range +/- 0.3..(as I read that somewhere?). The updateWeights() method is used to apply the weight changes calculated in the backwards pass of the backpropagation algorithm.

All of the file input/output is handled in the file NetworkApp. It is also the file that contains my programs main method. I use a series of scanners to read in all the parameters, input patterns, and expected outputs and store them in arrays. I had to use a separate scanner to read the number of lines in the input file in order to use arrays. I did this because I did not want to use ArrayLists as I find the syntax required to use them unwieldy, and they are slower than traditional arrays. I considered using a separate class for the file input/output in order to keep NetworkApp clean and used only for the interface and set up. I decided against this as it would have required passing information from some FileIO class to NetworkApp and then again to NeuralNetwork to actually initialise the network and that seemed to be overly complicated from an object oriented perspective.

Using the information from the input files I initialise a NeuralNetwork and then program execution goes into a while loop where operations can be specified by user input.

//Begin talking about NeuralNetwork.java here.