**AI Methods Coursework Part 2**

**Owen Read - F025172**

Introduction

The following report is going to describe and explain the methods I followed before, during and after programming the river flow neural network.

Chosen Language

For the development of the Neural network, I used java for the main program and python for the graphs plotting the rate at which the network learns. The reason I chose java was because of its fast runtime, meaning I can run a maximum of 9999999 epochs (due to matplotlib constraints) in a realistic time. For example, up to 10000 epochs takes less than a second to run. Python was my choice of graph plotting languages as the library matplotlib makes plotting graphs very easy and intuitive. The ANN is limited to 1 layer of hidden nodes but everything else is customisable.

Implementation Method

Whilst programming I used a procedural method. The main way of storing values was in 1D and 2D double arrays. Whilst programming I had to change from “Double” to “double” as I found that using the type ‘Double’ means the values are stored in heap memory which takes longer to fetch data from. Using these arrays I was able to do operations on each value by iterating through each of them using for loops and indexing e.g,: array[i][j].

My main program has 1 class that is just the whole neural network. The methods that is has include;

* getData(): gets the input data from a csv
* getDesiredData(): gets observed data value to use for training
* initWeights(): initialises starting weight values
* normaliseData(): puts the inputted data into a correct range and can also revert this process (input must be array)
* normaliseSingle(): Same as normaliseData but for a single double value
* wSum(): calculates weight sum from 2 arrays as parameters
* errorFunc(): calculates our values error
* getOverall(): gets sum of all the errors in an epoch
* backprop(): main backprop algorithm
* derivative(): gets value of x passed through the derivative of sigmoid
* feedforward(): main feedforward algorithm
* plotGraph(): writes all error values to txt for graph plotter code to create graph
* main(): runs combination of methods in correct order

Data pre-processing

Cleaning Data

The data set that was given came with some interesting outliers. Some presented were impossible values for the data that we are using such as negative Daily flow/ Daily rainfall. Another outlier that was presented were alphabetic entries such as ‘#’. To deal with those I combined a multiple thing such as:

* Removing the whole column
* Calculating the mean of the 2 values either side of where the outlier was

The reason I chose to go with these methods is because they continue to make the data legitimate for the Neural net to train from. If the value was numeric but is wrong the mean was taken as it could have been the user creating the data set accidentally pressing the value. The reason I wanted to delete values that had alphabetic characters as there was not a general idea to go from, meaning the original value has no relation to the dataset.

Standardising Data

To standardise the data I had to use a formula that allows us to make any value in a given range a value between 0-1 so that It can be accurately passed through sigmoid. To revert the value we would rearrange the formula.

Doing this was easy as I wrote a function in my java program that does this automatically for me.

Suitable Predictors

Initially I thought that there would be 7 inputs, as you could not use the Skelton value. However, after some more thinking I realised we can use the Skelton value from the day prior to predict one day ahead, this is because this value will be available for us as it would’ve been recorded at the end of the day. This means that the neural net has 8 input neurons.

Splitting Dataset

Splitting the dataset is done in my main java program. It will read the data from the file and add to an Array List which is then broken into 3 different sublists in the ration 60/20/20. The list that is 60% of the data set is used for training, the first 20% is used as the Validation set. The final 20% is used as the Test set.

Development

During development I began writing my code in python using mainly dictionaries. Upon completing the code I realised that the program was taking too long to run each set of epochs. From knowing this I decided to reprogram the Neural Network in java as the runtime in java is miniscule compared to Python. This allowed me to test and gather more results faster using the Java version of the program.

Whilst developing I had to think of a break case for the Neural Network so that it isn’t overtrained with the given testing set of data. To do this I had to outline when the training would break. I came up with a constant value of 0.005, meaning if the error for the previous epoch was less than 0.005 we would stop training. If the user inputted how many epochs to run, it would still break if the error was less than the value and output to the user how many epochs it took. If the error value is never met then the user will also be told this information and the final error of the last epoch.