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 $\frac{x+y}{2}$

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.

$$normalised\ x = \frac{x - minVal}{maxVal - minVal}$$

$$x = (normalised\ x * (maxVal - minVal)) + minVal$$

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 sub lists in the ratio 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.

Chosen Network

The structure of my Neural Network is 8-6-1. (8 Inputs, 6 Hidden, 1 Output). The reason for 8 inputs is we can input the 7 normal values and also use the value for skelton on the previous day to predict the value at skelton for the next day. Upon doing some research I quickly found out that there have been some studies behind using a certain formula for selecting the number of hidden nodes:

$$\frac{2(Number\ of\ Inputs)}{3} + Number\ of\ Outputs$$

Therefore, using this equation and the model of the network I am using 6 hidden Nodes. Only 1 output node is needed as we are only predicting one value, Skeltons mean daily flow.

Development

Python vs Java

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.

Overtraining Management

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. This is so that the network was not at danger of overtraining. 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.

Efficient Development

When developing the Neural network I had to think of a basic scenario in order to test that the maths behind my neural network works. Therefore I decided to program my network to work for a XOR gate. Meaning I was forced to program in a way that the number of nodes in each layer is not static so that I can easily modify it for the given Dataset.

Initialising Weights/Biases

I decided to initialise my weights by using a random value between 0-1 then dividing it by the number of neurons going into it. For example, for each hidden node you would divide by the Input Dimensions.

Since moving to this technique of initialisation I noticed instant improvement in the speed at which my neural net can train fully.

For the biases at each node, they are Initially given the value of 0.

User Interaction

When the program is booted the user is greeted with a question asking if they would like to;

- Train
- Run
- Predict
- Stop

Each of these answers performs a different action.

Train – This action is the main part of the program. The user which selected this can then input an amount of epochs to run. On the first run the program will break if the epoch error is less than 0.005 or when the epochs have been completed, the user will be notified of which case it is. After this the user can chose to continue training (will have to input more epochs and a new epoch error to break at), or to run the neural net or stop the program.

Run – This uses the Validation set and will output to the user all the inputs with the predicted output alongside the desired output. This way the user can visually see how close the Neural net is to predicting values. Along with this the Program will output the % efficiency of the Neural net based on the validation set. To do this it uses the formulas shown below:

$$\%\, Error = \; 100 - \frac{\left(\sum_i \frac{|Observed_i - Predicted_i|}{Observed_i} * 100\right)}{n}$$

This allowed me to understand how accurate my neural network was getting.

Predict – When the user selected this the program will predict the value for the day next day using the test set of data. Predict will display to the user the value it predicts for Skeltons next day.

Stop – This will end the program for the user

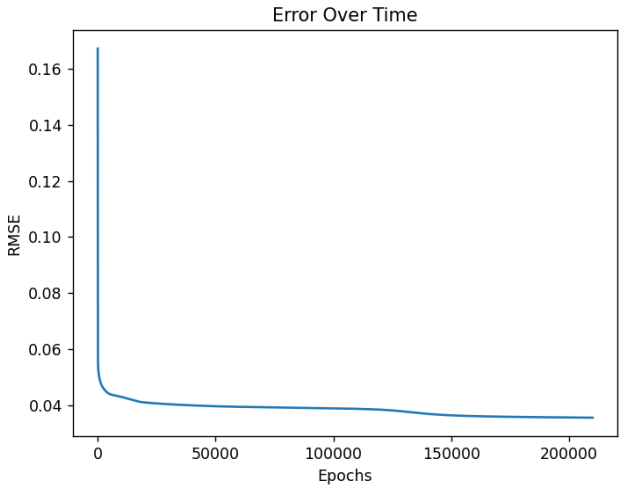
My python file has little user interaction. This program just requires the user to run the file, after this the program is automatically run to create a line graph with RMSE (y Axis) against Epoch Number (x Axis):

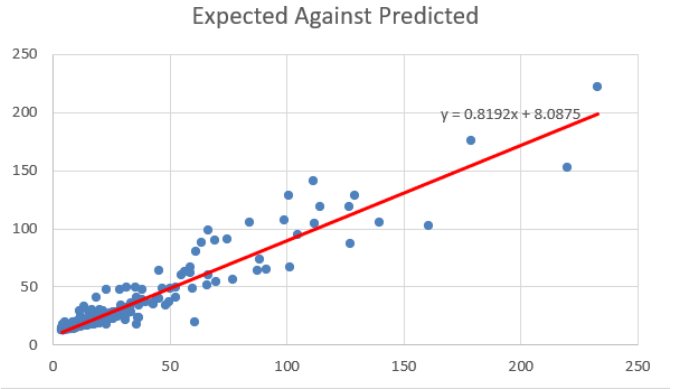
$$RMSE(x,y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Graph Plotting

My program currently has the capabilities of plotting 2 types of graphs. One being a line graph of RMSE against epochs and the other being a scatter graph of Expected value against predicted Value. For both it writes the correct values to a txt file which is accessed by the python graph plotting file I created. The python will either plot the graph or write the new values in an excel file for the user to create the graph.

The graphs will look something like this:





The graphs will be evaluated further on into the report.

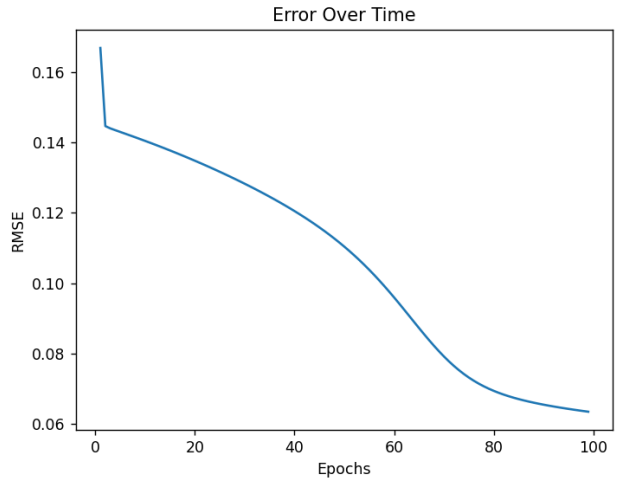
Improvements to standard Backpropagation

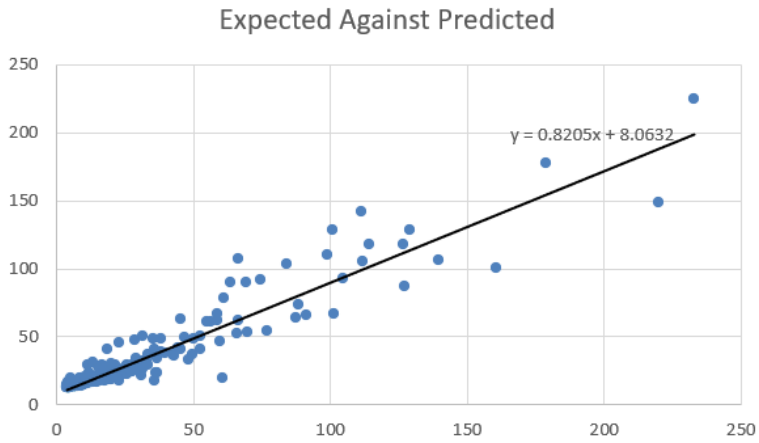
Bold Driver

Momentum

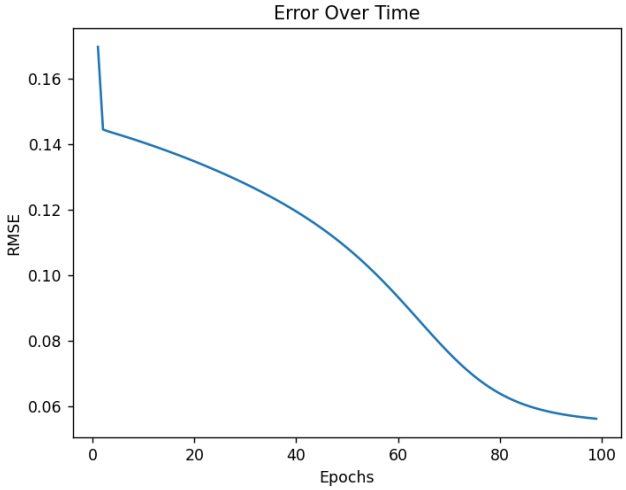
Weight Decay

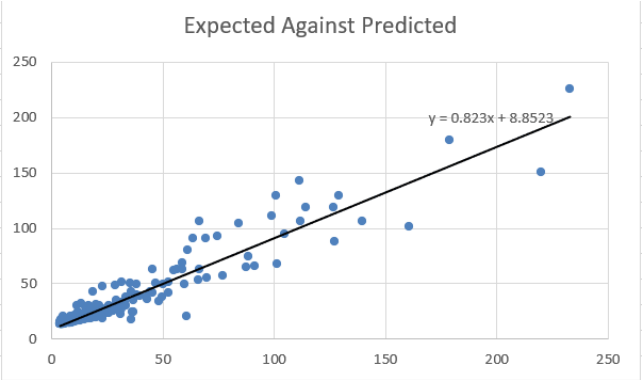
With decay





Without decay





Weight decay is used to avoid overfitting the training data therefore results will be better with the validation set and training set. Weight decay can also avoid the exploding gradient problem. This is where the error gradients can accumulate during an update and result in large gradients. This in return causes large changes to the network weights making an unstable network.

By looking closely, you can see the small curve in the graph without weight decay is a little more aggressive than the one with weight decay. This shows that the weight decay is preventing large changes to the network.

Using the scatter graphs, we can see the gradient of both graphs. With the one not using weight decay being 0.823, and the other being 0.8205. A perfect neural network model would have a value of 1. We can see that not using decay in this circumstance is more correlated to the data. This however is over a small number of epochs (10,000). As more epochs occur the risk of overtraining becomes greater. This is where weight decay kicks in and will outperform the network without weight decay.

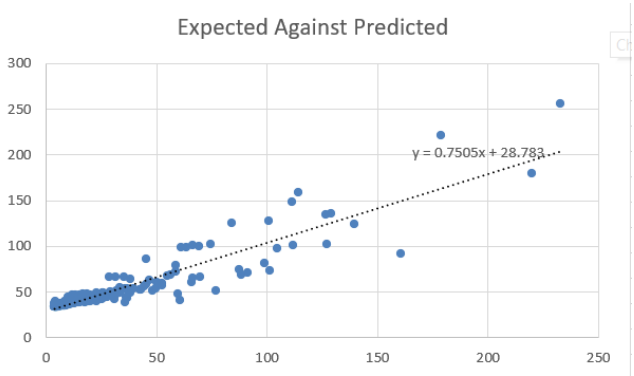
Annealing

Predicting value

The task was to predict the Mean Daily Flow of Skelton 1 day in advance. My neural network can produce a value to some extent of accuracy shown below (Using only weight decay):

Epoch	Value Predicted	% Accuracy Of Model	RMSE
100	48.934	-149.69%	0.061403
1000	37.059	-42.20%	0.049255
5000	34.877	11.31%	0.044120
10000	36.344	19.54%	0.042881
100000	37.983	59.15%	0.037834
500000	41.980	72.89%	0.033480
1000000	41.409	71.47%	0.033094

From this experiment we can see that the model started to overtrain as the Accuracy of the model decreased from the previous epochs. Using initiative, I would guess the correct predicted value to be around 40.



Comparison Between Alternative Data Driven Model

The data driven model I have chosen to compare against is using the LINEST function within excel. This function will work out the m values of each set of x values to work out the average trend.

With this data we can apply it to the function to get a predicted output:

$$Prediction = \left(\sum_i x_i\, m_i\right) + y\ intercept$$

The table that is produced is shown below.

	Snaize	Malham	East Cow	Arkeng	Skelton	Westwick	Skip Bridge	Crakehill	y Intercept
m Val	0.418484	0.088448958	0.079314	0.263055	0.360626754	0.594221517	1.21539685	-0.066303556	2.12910382
	0.033032	0.026136526	0.048257	0.052486	0.043667962	0.044957318	0.152159811	0.08557747	0.63969995
	0.903364	17.35155843	#N/A	#N/A					
	1692.016	1448	#N/A	#N/A					
	4075410	435958.888	#N/A	#N/A					

When passing in the values of the last row of the data set to predict the next value we get:

Crakehill	Skip Bridge	Westwick	Skelton	Arkengarthdale	East Cowton	Malham Tarn	Snaizeholme	Predicted
0.8	6.4	9.6	4.8	39.42	13.549	5.829	16.8	36.2805039

Comparing this to the values we got from the neural network (39.76) in the earlier explained table we can see that the average of those obtained values is rather close to the statistical prediction model. We must also keep to mind that the average of the neural net obtained values is a bit skewed as the overtrained and undertrained values were also included. If these were not included the average would be 37.75 which is much closer to the value that the statistical model got.

Conclusion

In conclusion, all the changes I made to my neural network was to increase the speed at which it was adequately trained. One change to do this was using 6 hidden nodes. Where less hidden nodes took too long.

Another change was weight decay so that it would not allow my network to get overtrained as easily, meaning the overfitting and the exploding gradient would not be an issue after a large amount of epochs.

My neural network (8-6-1) was reliably able to predict values after many epochs. Somewhere in the range of 100,000 – 500,000 was where the network got to 70% accuracy.

SOURCE CODE

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.IntStream;

public class neuralNetwork{

    // Initialising Variables

    public static int I_dim = 8;
    public static int H_dim = 8; //5
    public static int O_dim = 1;

    public static int epochCount = 4;
    public static Double learning_param = 0.1;
```

```
    public static double minVal = 3.694; //0; Change this depending on data
set
    public static double maxVal = 448.1; //1; Change this depending on data
set

    public static double[][] weightToHid = new double[H_dim][I_dim];
    public static double[] weightToOut = new double[H_dim];
    public static double[] hidBias = new double[H_dim];
    public static double[] hidVals = new double[H_dim];
    public static double[] hidSelfWeight = new double[H_dim];
    public static double[] outSelfWeight = new double[O_dim];
    public static double outBias = 0;
    public static double outVal = 0.0;
    public static double[] outDelta = new double[O_dim];
    public static double[] hidDelta = new double[H_dim];

    public static ArrayList<Double> dotExpected = new ArrayList<Double>();
    public static ArrayList<Double> dotGot = new ArrayList<Double>();

    //XOR Testing
    //public static double[][] data = {{0.0, 0.0}, {0.0, 1.0}, {1.0, 0.0},
{1.0, 1.0}}; //getData("data");
    public static double[] desiredOut = getDesiredData("training"); //{0.0,
1.0, 1.0, 0.0};

    public static double[][] data = getData("training");
    public static double[][] validation = getData("validation");
    public static double[][] test = getData("test");

    // Used to convert 2D ArrayLists into 2D Double

    public static double[][] convertArrayList(ArrayList<String[]> theList){
        double[][] result = new double[theList.size()][8];
        int size = theList.size();
        for (int i = 0; i < size; i++){
            int size3 = theList.get(i).length;
            double[] temp = new double[size3];
            for (int y = 0; y < size3; y++){
                if (theList.get(i)[y] != ""){
                    temp[y] = Double.parseDouble(theList.get(i)[y]);
                }
            }

            result[i] = temp;
        }
        return result;
    }
```

```
    //Used to convert 1D ArrayLists into 1D Double Lists

    public static double[] convert1DArrayList(ArrayList<String> theList){
        double[] result = new double[theList.size()];
        int size = theList.size();
        for (int i = 0; i < size; i++){
            result[i] = Double.parseDouble(theList.get(i));
        }
        return result;
    }

    //Reads data from csv and puts into array for later use

    public static double[][] getData(String option){
        //XOR
        //double[][] theData = {{0.0, 0.0}, {0.0, 1.0}, {1.0, 0.0}, {1.0,
1.0}};
        try {
            ArrayList<String[]> theData1 = new ArrayList<String[]>();
            BufferedReader csvReader = new BufferedReader(new
FileReader("DataSet.csv"));
            String row;
            int x = 0;
            while ((row = csvReader.readLine()) != null) {
                if (x > 1){
                    String[] data = row.split(",");
                    theData1.add(Arrays.copyOfRange(data, 1, data.length));
                } else {x++;}
            }

            theData1.remove(theData1.size() - 1);

            csvReader.close();
            double i1 = Math.round(theData1.size() * 0.6);
            double i2 = i1 + Math.round(theData1.size() * 0.2);

            ArrayList<String[]> trainingSet1 = new ArrayList<String[]>();
            ArrayList<String[]> ValidationSet1 = new ArrayList<String[]>();
            ArrayList<String[]> TestSet1 = new ArrayList<String[]>();

            for (int i = 0; i < theData1.size(); i++){
                if (i < i1){
                    trainingSet1.add(theData1.get(i));
                }
                if (i1 <= i && i < i2){
                    ValidationSet1.add(theData1.get(i));
                }
                if (i >= i2){
```

```
                    TestSet1.add(theData1.get(i));
                }
            }

            double[][] trainingSet = convertArrayList(trainingSet1);
            double[][] validationSet = convertArrayList(ValidationSet1);
            double[][] testSet = convertArrayList(TestSet1);


            if (option.equalsIgnoreCase("training")){
                return trainingSet;
            }
            if (option.equalsIgnoreCase("validation")){
                return validationSet;
            }
            if (option.equalsIgnoreCase("test")){
                return testSet;
            }



        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return data;

    }

    //Reads data from csv and puts into array for later use
    public static double[] getDesiredData(String option){
        //XOR
        //double[][] theData = {{0.0, 0.0}, {0.0, 1.0}, {1.0, 0.0}, {1.0,
1.0}};
        try {
            ArrayList<String> desiredOut1 = new ArrayList<String>();
            BufferedReader csvReader = new BufferedReader(new
FileReader("DataSet.csv"));
            String row;
            int x = 0;
            while ((row = csvReader.readLine()) != null) {
                if (x > 1){
                    String[] data = row.split(",");
                    desiredOut1.add(data[4]);
                } else {x++;}
            }

            desiredOut1.remove(0);
```

```
            csvReader.close();

            ArrayList<String> trainingOut = new ArrayList<String>();
            ArrayList<String> ValidationOut = new ArrayList<String>();
            ArrayList<String> TestOut = new ArrayList<String>();

            double i1 = Math.round(desiredOut1.size() * 0.6);
            double i2 = i1 + Math.round(desiredOut1.size() * 0.2);

            for (int i = 0; i < desiredOut1.size(); i++){
                if (i < i1){
                    trainingOut.add(desiredOut1.get(i));
                }
                if (i1 <= i && i < i2){
                    ValidationOut.add(desiredOut1.get(i));
                }
                if (i >= i2){
                    TestOut.add(desiredOut1.get(i));
                }
            }

            if (option.equalsIgnoreCase("training")){
                return convert1DArrayList(trainingOut);
            }
            if (option.equalsIgnoreCase("validation")){
                return convert1DArrayList(ValidationOut);
            }
            if (option.equalsIgnoreCase("test")){
                return convert1DArrayList(TestOut);
            }


        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return desiredOut;

    }

    //public static double[][] data = getData();
    //public static double[] desiredOut = getDesiredData();

    //Initialises Weight values at first run

    public static void initWeights(){
        int max = 1;
        int min = -1;
        for (int i = 0; i < H_dim; i++){
```

```
            for (int x = 0; x < I_dim; x++){
                weightToHid[i][x] = Math.random() / I_dim;
            }
        }

        for (int i = 0; i < H_dim; i++){
            weightToOut[i] = ((Math.random() * (max - min)) + min) / H_dim;
            hidBias[i] = 0;
            hidVals[i] = ((Math.random() * (max - min)) + min) / I_dim;
            hidSelfWeight[i] = ((Math.random() * (max - min)) + min) / I_dim;
            hidDelta[i] = 0.0;
        }

        outSelfWeight[0] = Math.random() / H_dim;
        outDelta[0] = 0.0;
    }

    // stadardises data between 0-1 range
    // Can also de standardise (In ARRAYS)
    public static double[] normaliseData(double[] data, String thisCase){

        if (thisCase == "pre"){
            double[] answer = new double[data.length];
            int size = answer.length;
            for (int i = 0; i < size; i++){
                answer[i] = (data[i] - minVal) / (maxVal - minVal);
            }
            return answer;

        } else {
            double[] answer = new double[data.length];
            int size = answer.length;
            for (int i = 0; i < size; i++){
                answer[i] = (data[i] * (maxVal - minVal)) + minVal;
            }
            return answer;
        }


    }

    // stadardises data between 0-1 range
    // Can also de standardise (In ARRAYS)

    public static double normaliseSingle(double data, String thisCase){

        if (thisCase == "pre"){
            double answer;
```

```
            answer = (data - minVal) / (maxVal - minVal);
            return answer;

        } else {
            double answer;
            answer = (data * (maxVal - minVal)) + minVal;
            return answer;
        }
    }

    // Calculates weighted sum of 2 arrays
    public static double wSum(double[] m1, double[] m2){
        double sum = 0.0;
        int size = m1.length;
        for (int i = 0; i < size; i++){
            sum += m1[i] * m2[i];

        }
        return sum;
    }

    // Calculates our error value
    public static double errorFunc(double predicted, double real){
        return Math.pow((predicted - real), 2);
    }

    // Activation function (Sigmoid)
    public static double activation(double x){
        //Sigmoid
        double result = 1 / (1 + Math.exp(-x));
        return result;
    }

    // Gets sum of an array
    public static double getOverall(double [] thisData){
        double sum = 0.0;
        int size = thisData.length;
        for (int i = 0; i < size; i++){
            sum += thisData[i];
        }
        return sum;
    }

    // Gets omega value for weight decay
    public static double getOmega(){
        int totalWeights = (H_dim * O_dim) + (H_dim * I_dim) + H_dim + O_dim;
        double weightSquared = 0.0;
        for (int i = 0; i < H_dim; i++){
```

```
            weightSquared += Math.pow(weightToOut[i], 2);
            for (int j = 0; j < I_dim; j++){
                weightSquared += Math.pow(weightToHid[i][j], 2);
            }
        }
        return weightSquared / (2 * totalWeights);

    }

    // Back prop algorithm
    public static void backProp(int dataIndex, double observed, double[][]
thisData, int currentEpoch){
        // Values needed
        double desiredOutVal = normaliseSingle(observed, "pre");
        double[] inpVal = thisData[dataIndex];
        double upsilon = 1 / (learning_param * currentEpoch+1);
        double omega = getOmega();
        double upsilonOmega = upsilon * omega;

        // Calculates delta values and weight changes
        for (int i = 0; i < O_dim; i++){
            outDelta[i] = (desiredOutVal - outVal + upsilonOmega) *
derivative(outVal);
            outBias += (learning_param * outDelta[i]);
            for (int x = 0; x < H_dim; x++){
                weightToOut[x] += (learning_param * outDelta[i] * hidVals[x]);
                hidDelta[x] = (weightToOut[x] * outDelta[i]) *
derivative(hidVals[x]);
                hidBias[x] += (learning_param * hidDelta[x]);
                for (int j = 0; j < I_dim; j++){
                    weightToHid[x][j] += (learning_param * hidDelta[x] *
normaliseSingle(inpVal[j], "pre"));
                }
            }
        }

    }

    // Derivative of sigmoid
    private static double derivative(double x) {
        return (x * (1 - x));
    }

    //Feed forward algorithm

    public static double[] feedForward(int epochs, boolean training,
double[][] useThisData, double[] useTheseOutputs, double breakCase) throws
IOException{
```

```
        double[] epochErrors = new double[epochs];
        double[] results = new double[useThisData.length+1];
        double[][] trainingData = getData("training");
        for (int j = 0; j < epochs; j++){
            double[] errors = new double[useThisData.length];

            for (int i = 0; i < useThisData.length; i++){
                double[] thisPass = useThisData[i];
                for (int x = 0; x < H_dim; x++){
                    double[] weights = weightToHid[x];

                    double[] norm = normaliseData(thisPass, "pre");
                    double wS = wSum(weights, norm) + (hidSelfWeight[x] *
hidVals[x]);

                    hidVals[x] = activation(wS);
                }

                for (int x = 0; x < O_dim; x++){
                    double wS = wSum(weightToOut, hidVals) + (outSelfWeight[x]
* outVal);
                    outVal = activation(wS);

                    double error = errorFunc(outVal,
normaliseSingle(useTheseOutputs[i], "pre"));

                    //String results = String.format("Expected:
%f             Got: %f", normaliseSingle(useTheseOutputs[i], "pre"), outVal);
                    //System.out.println(results);
                    errors[i] = error;
                }

                if (training){
                    backProp(i, useTheseOutputs[i], trainingData, j);
                } else {
                    results[i] = normaliseSingle(outVal, "post");
                    dotGot.add(normaliseSingle(outVal, "post"));
                    dotExpected.add(useTheseOutputs[i]);
                }

            }
            epochErrors[j] = Math.pow(getOverall(errors) / useThisData.length,
0.5);

            if (epochErrors[j] < breakCase && training){
                System.out.println(String.format("Successfuly Trained in %d
Epochs", j));
                return(epochErrors);
```

```
            }
        }

        if (training){
            System.out.println(String.format("Epoch Amount Hit Current Error:
%f", epochErrors[epochErrors.length-1]));
            return(epochErrors);
        } else {
            return results;
        }
    }

    // Writes values to txt file for graph plotting
    public static void plotErrorGraph(ArrayList<Double> errorResults) throws
IOException{
        new FileWriter("errorData.txt").close();

        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("errorData.txt"))){
            for (double line: errorResults){
                writer.write (line + "\n");
            }

            writer.close();
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }

    // Writes values to txt file for graph plotting
    public static void plotDot(ArrayList<Double> expected, ArrayList<Double>
got) throws IOException{
        new FileWriter("dotGraph.txt").close();

        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("dotGraph.txt"))){
            for (int i = 0; i < expected.size(); i++){
                writer.write(String.format("%f,%f\n", expected.get(i),
got.get(i)));
            }

            writer.close();
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }

    // Gets accuracy of the current model
```

```
    public static double getAccuracy(double[] thisTable, double[] desired) {
        double sum = 0.0;

        for (int i = 0; i < thisTable.length - 1; i++){
            double top = (Math.abs(desired[i] - thisTable[i]) / desired[i]) *
100;
            sum += top;
        }
        double percent = 100 - (sum / thisTable.length - 1);

        return percent;

    }

    // Creates user UI and runs code in correct Order
    public static void main(String[] args) throws IOException{

        /*
        double[][] theData = getData("validation");
        double[] theResults = getDesiredData("validation");

        for (int i = 0; i < theData.length; i++){
            System.out.println(String.format("%.2f, %.2f, %.2f, %.2f, %.2f,
%.2f, %.2f, %.2f, %.2f", theData[i][0], theData[i][1], theData[i][2],
theData[i][3], theData[i][4], theData[i][5], theData[i][6], theData[i][7],
theResults[i]));
        }*/



        ArrayList<Double> errorResults = new ArrayList<Double>();

        System.out.println("Would you like to: TRAIN the AI, RUN the AI,
predict the next value or STOP the program");
        Scanner userInp = new Scanner(System.in);
        String accInp = userInp.nextLine();

        while (!accInp.equalsIgnoreCase("stop")){

            if (accInp.equalsIgnoreCase("train")){
                initWeights();
                System.out.println("How Many Epochs");
                Scanner epochInp = new Scanner(System.in);
                int epoch = epochInp.nextInt();

                double[] tempVal = feedForward(epoch, true,
getData("training"), getDesiredData("training"), 0.005);
```

```
                IntStream.iterate(0, i -> i + 1).limit(tempVal.length-
1).forEach(i -> errorResults.add(tempVal[i]));

                System.out.println("Training Completed");
            }

            if (accInp.equalsIgnoreCase("run")){

                double[][] validation = getData("validation");
                double[] desired = getDesiredData("Validation");

                double[] thisTable = feedForward(1, false, validation,
desired, 0.05);


                IntStream.iterate(1, i -> i + 1).limit(I_dim).forEach(i ->
System.out.print(String.format("   Input%d    |", i)));
                System.out.print("   Expected    |");
                System.out.print("   Result    |");
                System.out.println();
                IntStream.iterate(0, i -> i + 1).limit(I_dim + 2).forEach(i ->
System.out.print("--------------"));
                System.out.println("--");
                IntStream.iterate(0, i -> i + 1).limit(validation.length-
1).forEach(i ->
System.out.println(String.format("   %.1f   |   %.1f   |   %.1f   |   %.1f   |
  %.1f   |   %.1f   |   %.1f   |   %.1f   |   %.6f   |   %.6f   |",
data[i][0], data[i][1], data[i][2], data[i][3], data[i][4], data[i][5],
data[i][6], data[i][7], desired[i], thisTable[i])));
                //XOR IntStream.iterate(0, i -> i +
1).limit(data.length).forEach(i ->
System.out.println(String.format("   %.1f   |   %.1f   |   %.6f   |   %.6f   |
", data[i][0], data[i][1], desiredOut[i], thisTable[i])));

                double accuracy = getAccuracy(thisTable, desired);

                System.out.println(String.format("The accuracy of the model is
%.2f%s", accuracy, "%"));

                System.out.println("Would you like to continue training until
a new error threshold is met (1st run is 0.05). If so please enter a value or
type no to cancel: ");
                Scanner errorVal = new Scanner(System.in);
                String accErrorVal = errorVal.nextLine();

                if (!accErrorVal.equalsIgnoreCase("no")){
                    System.out.println("Please enter epoch count");
                    Scanner epochInp = new Scanner(System.in);
                    int accEpochInp = Integer.parseInt(epochInp.nextLine());
```

```
                    double[] tempVal = feedForward(accEpochInp, true,
getData("training"), getDesiredData("training"),
Double.parseDouble(accErrorVal));
                    IntStream.iterate(0, i -> i + 1).limit(tempVal.length-
1).forEach(i -> errorResults.add(tempVal[i]));
                }
            }

            if (accInp.equalsIgnoreCase("predict")){
                feedForward(1, false, getData("test"), getDesiredData("test"),
0.005);
                System.out.println(normaliseSingle(outVal, "post"));
            }

            System.out.println("Would you like to: TRAIN the AI, RUN the AI,
predict the next value or STOP the program");
            userInp = new Scanner(System.in);
            accInp = userInp.nextLine().toLowerCase();
        }
        plotErrorGraph(errorResults);
        plotDot(dotExpected, dotGot);
    }
}
```

```
import matplotlib.pyplot as mp
import numpy as np
from openpyxl import Workbook


def plotError():
    data = []
    with open("errorData.txt", "r") as file:
        for eachLine in file:
            if (eachLine != "0.0\n"):
                data.append(float(eachLine))

    lst = list(np.arange(1,len(data)+1))
    mp.plot(lst, data)
    mp.xlabel("Epochs")
    mp.ylabel("RMSE")
    mp.title("Error Over Time")
    mp.show()

def plotDot():
    x = []
    y = []
    with open("dotGraph.txt", "r") as file:
        for eachLine in file:
```

```
            if (eachLine != "0.0\n"):
                data = eachLine.split(",")
                x.append(data[0])
                y.append(data[1])

    wb = Workbook()
    ws = wb.active

    ws["A1"] = "x"
    ws["B1"] = "y"
    for i in range(1, len(x)+1):
        ws['A%s'%str(i+1)] = float(x[i-1])
        ws['B%s'%str(i+1)] = float(y[i-1])

        wb.save("dotGraph.xlsx")


plotDot()
```