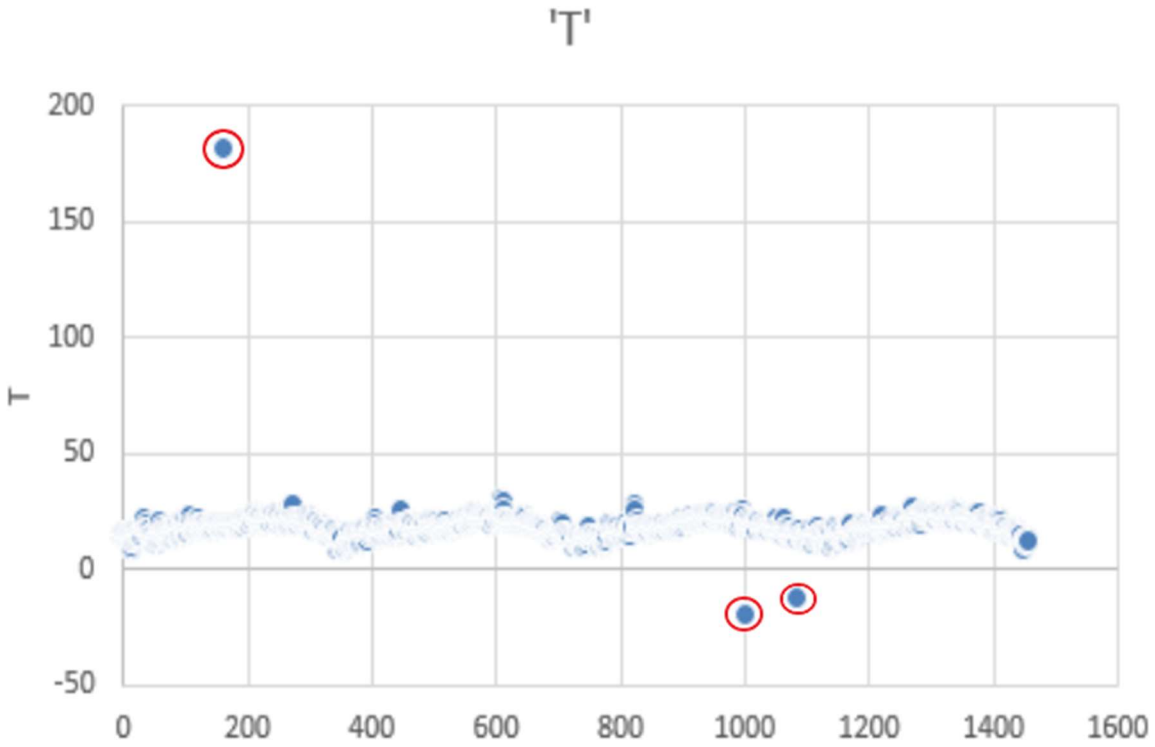
Data Pre-Processing

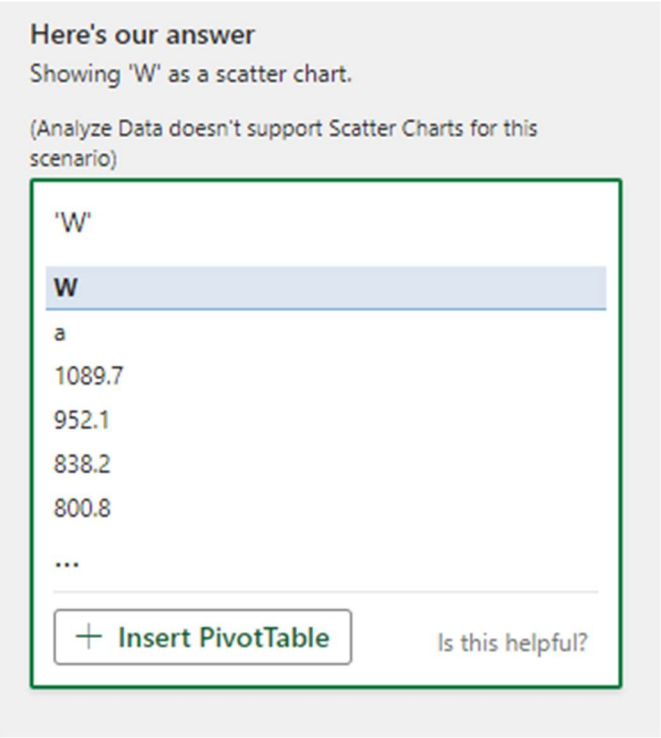
Cleansing

For when I standardise the data, it makes it easier if only the essential information is there, I therefore then also the entire date column as that column was purely there for reference and is not a predictor, nor a predictand.

I then used the “analyse data” section on Microsoft excel to plot each of the individual columns, with respect to its row number.



e.g. For temperature, the values labelled in red are clearly way out of reach of the general pattern, so I deleted the entire row associated with that value for each of them.



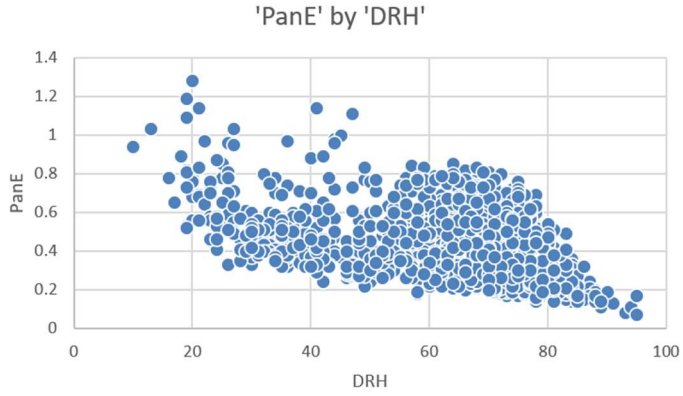
Whenever excel wasn't able to display the data, it was usually due to an invalid cell type in one of the rows of that column.

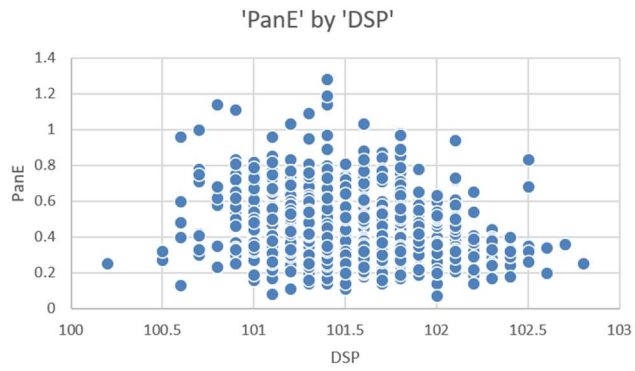
So in the above example, an "a" was in one of these rows, so I found the row that the "a" was on and deleted the entire row as it has missing data, and since our data is not time series, we cannot interpolate and therefore must remove the entire row

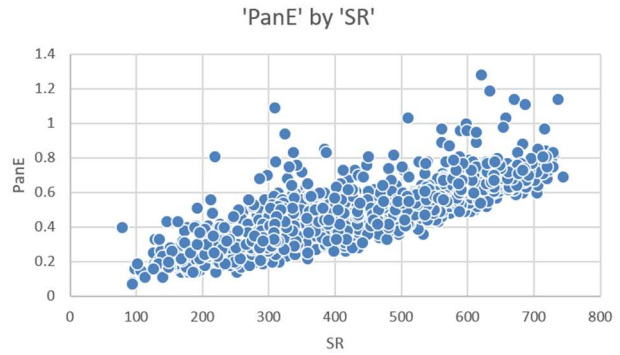
Identifying Correlations

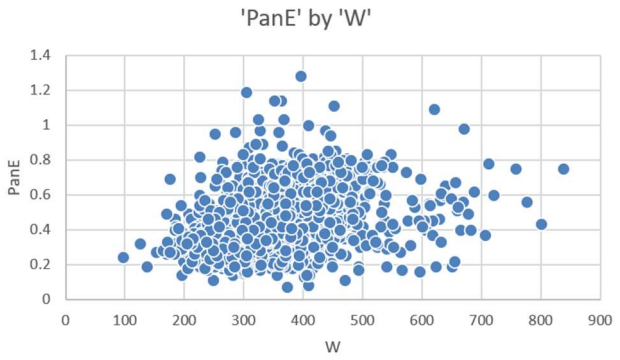
After cleansing the data, I needed to analyse it to figure out which inputs were necessary to use for the algorithm, and which ones I could try leaving out and experimenting whether that gives me more accurate predicting data

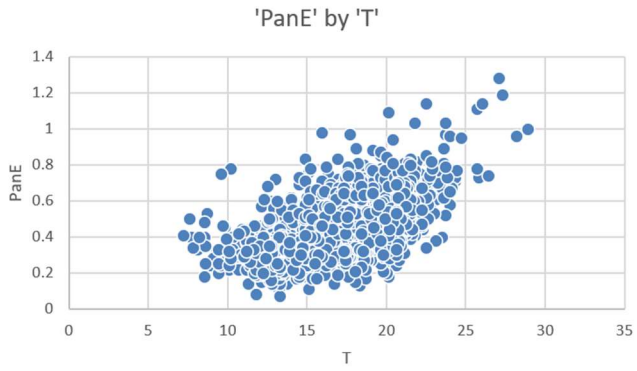
So I used the Analyse data feature on excel to find each of the scatter correlations for each predictor against the predictand PanE











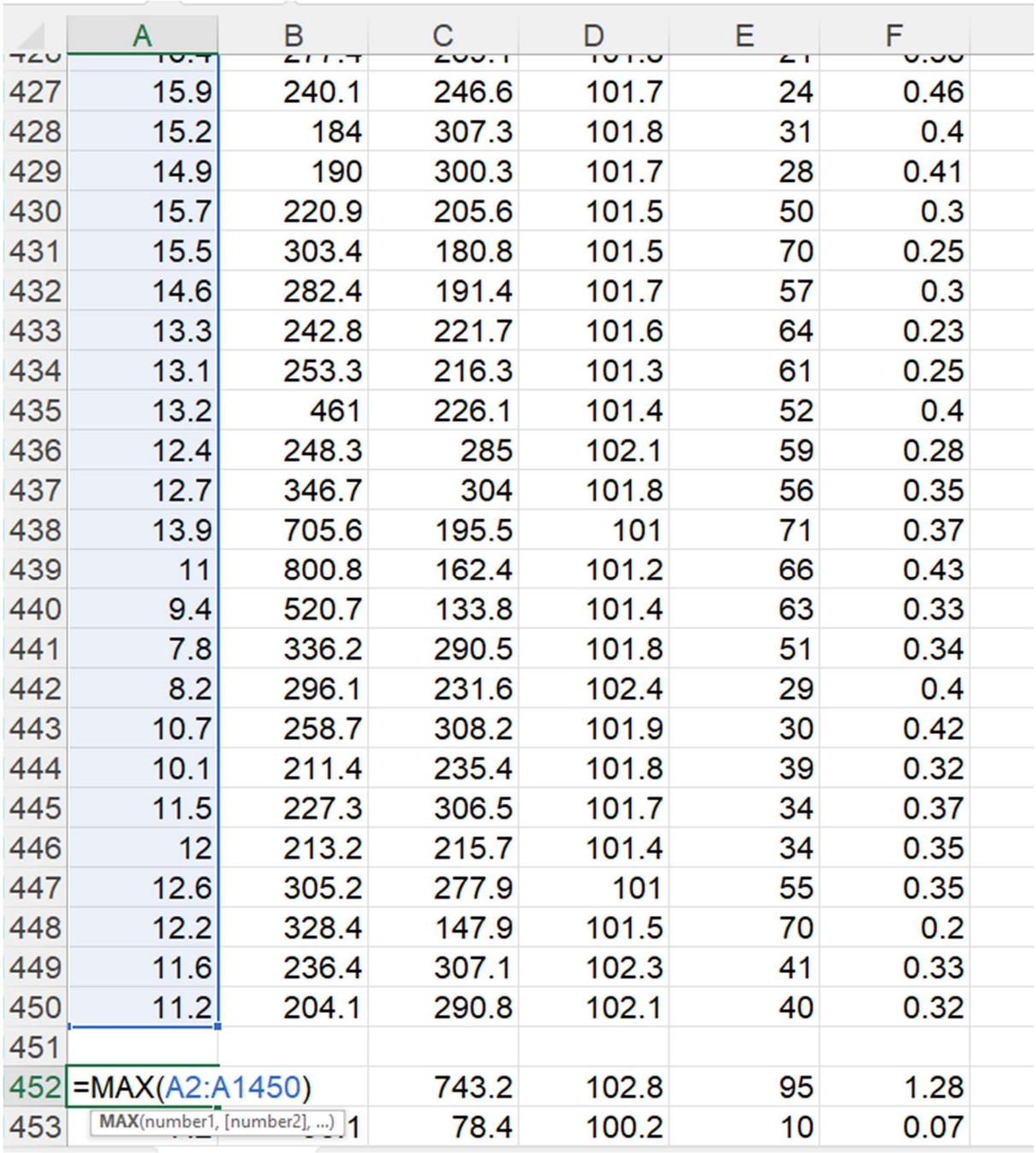
I noticed a strong correlation between each of the predictors, besides DSP, which seems significantly weaker.

So during my implementation, I will perform training for both all the data inputs, with and without DSP and investigate whether or not it gives me a lower mean squared error.

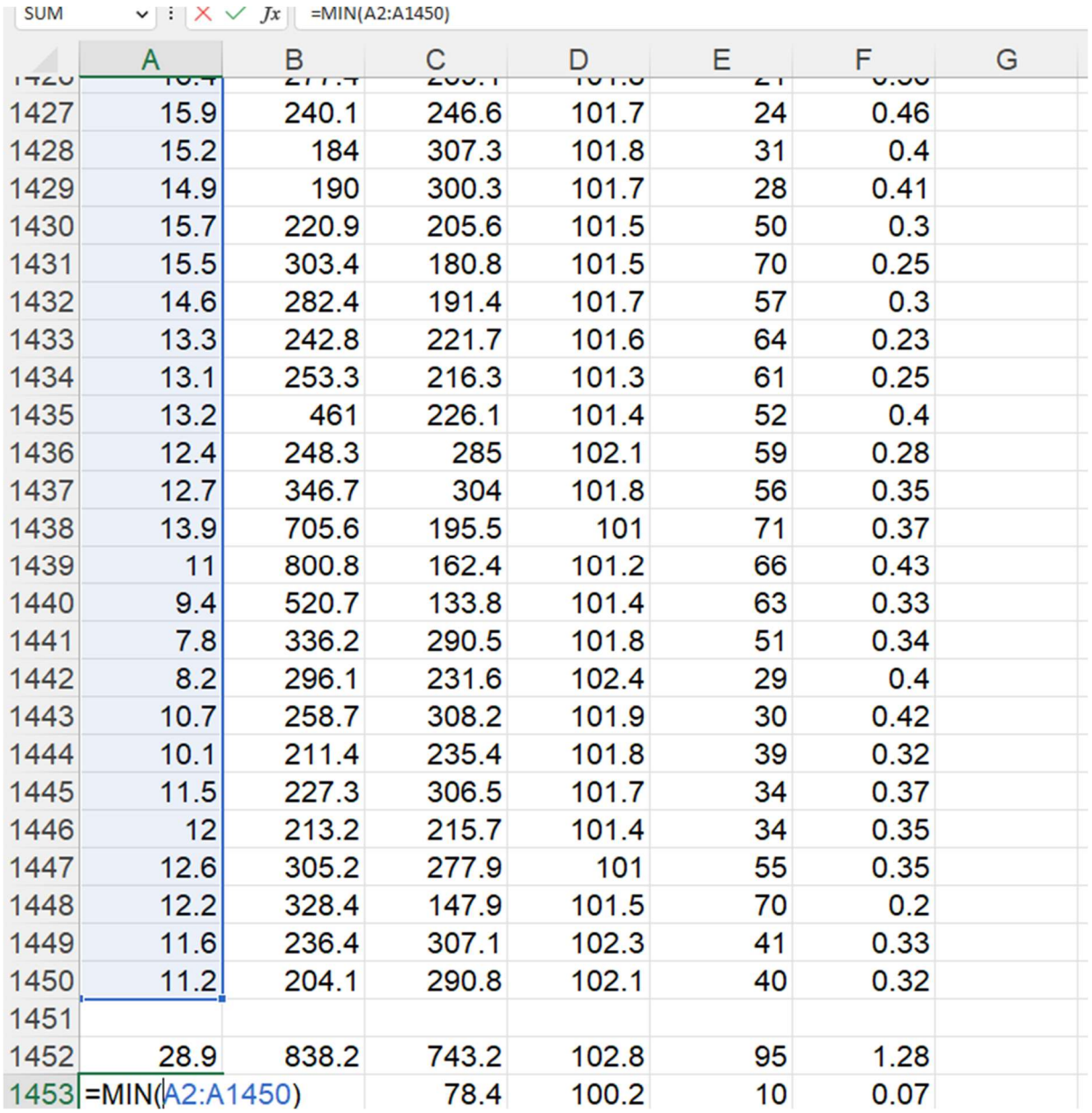
Standardisation

Once all the outliers and invalid data were removed, I needed to standardise the data

To do this, I first gathered each of the min and max values of each columns in the data set and saved the values elsewhere for later use when I plug each cell into the standardisation formula



	A	B	C	D	E	F
427	15.9	240.1	246.6	101.7	24	0.46
428	15.2	184	307.3	101.8	31	0.4
429	14.9	190	300.3	101.7	28	0.41
430	15.7	220.9	205.6	101.5	50	0.3
431	15.5	303.4	180.8	101.5	70	0.25
432	14.6	282.4	191.4	101.7	57	0.3
433	13.3	242.8	221.7	101.6	64	0.23
434	13.1	253.3	216.3	101.3	61	0.25
435	13.2	461	226.1	101.4	52	0.4
436	12.4	248.3	285	102.1	59	0.28
437	12.7	346.7	304	101.8	56	0.35
438	13.9	705.6	195.5	101	71	0.37
439	11	800.8	162.4	101.2	66	0.43
440	9.4	520.7	133.8	101.4	63	0.33
441	7.8	336.2	290.5	101.8	51	0.34
442	8.2	296.1	231.6	102.4	29	0.4
443	10.7	258.7	308.2	101.9	30	0.42
444	10.1	211.4	235.4	101.8	39	0.32
445	11.5	227.3	306.5	101.7	34	0.37
446	12	213.2	215.7	101.4	34	0.35
447	12.6	305.2	277.9	101	55	0.35
448	12.2	328.4	147.9	101.5	70	0.2
449	11.6	236.4	307.1	102.3	41	0.33
450	11.2	204.1	290.8	102.1	40	0.32
451						
452	=MAX(A2:A1450)		743.2	102.8	95	1.28
453	MAX(number1, [number2], ...)	1	78.4	100.2	10	0.07



	A	B	C	D	E	F	G
1427	15.9	240.1	246.6	101.7	24	0.46	
1428	15.2	184	307.3	101.8	31	0.4	
1429	14.9	190	300.3	101.7	28	0.41	
1430	15.7	220.9	205.6	101.5	50	0.3	
1431	15.5	303.4	180.8	101.5	70	0.25	
1432	14.6	282.4	191.4	101.7	57	0.3	
1433	13.3	242.8	221.7	101.6	64	0.23	
1434	13.1	253.3	216.3	101.3	61	0.25	
1435	13.2	461	226.1	101.4	52	0.4	
1436	12.4	248.3	285	102.1	59	0.28	
1437	12.7	346.7	304	101.8	56	0.35	
1438	13.9	705.6	195.5	101	71	0.37	
1439	11	800.8	162.4	101.2	66	0.43	
1440	9.4	520.7	133.8	101.4	63	0.33	
1441	7.8	336.2	290.5	101.8	51	0.34	
1442	8.2	296.1	231.6	102.4	29	0.4	
1443	10.7	258.7	308.2	101.9	30	0.42	
1444	10.1	211.4	235.4	101.8	39	0.32	
1445	11.5	227.3	306.5	101.7	34	0.37	
1446	12	213.2	215.7	101.4	34	0.35	
1447	12.6	305.2	277.9	101	55	0.35	
1448	12.2	328.4	147.9	101.5	70	0.2	
1449	11.6	236.4	307.1	102.3	41	0.33	
1450	11.2	204.1	290.8	102.1	40	0.32	
1451							
1452	28.9	838.2	743.2	102.8	95	1.28	
1453	=MIN(A2:A1450)		78.4	100.2	10	0.07	

```
ArrayList<ArrayList<Double>> tableData = new ArrayList<>();
try {
    FileInputStream fis = new FileInputStream(new File("DataSet.xlsx"));
    XSSFWorkbook wb = new XSSFWorkbook(fis);
    XSSFSheet sheet = wb.getSheetAt(0);
    for (Row row : sheet) {
        ArrayList<Double> rowData = new ArrayList<>();
        // start from the second cell
        for (Cell cell: row) {
            if (cell.getCellType() == Cell.CELL_TYPE_NUMERIC) {
                rowData.add(cell.getNumericCellValue());
            }
        }
        tableData.add(rowData);
    }
} catch(IOException e) {
    // handle the exception
}
```

I pushed all the data onto a 2d ArrayList of doubles

```
double max[] = {28.9,952.1,743.2,102.8,95,1.28};
double min[] = {7.2,96.1,78.4,100.2,10,0.07};
```

I stored the min and max value for each row in a min and max array

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        double newVal = 0.8*((tableData.get(i).get(j)-min[j])/(max[j]-min[j]))+0.1;
        tableData.get(i).set(j,newVal);
    }
}
```

For each of those cells, I used the standardisation formula for converting the data into the [0.1,0.9] format and replaced the cell in the ArrayList with the new value

```
public static void upload(ArrayList<ArrayList<Double>> data) throws FileNotFoundException, IOException{
    // Create a new workbook and sheet
    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet sheet = workbook.createSheet("Sheet1");

    // Iterate over the data and create rows and cells in the sheet
    int rowIndex = 0;
    for (ArrayList<Double> rowData : data) {
        Row row = sheet.createRow(rowIndex++);
        int cellIndex = 0;
        for (double cellData : rowData) {
            Cell cell = row.createCell(cellIndex++);
            cell.setCellValue(cellData);
        }
    }

    // Write the workbook to an Excel file
    FileOutputStream fileOut = new FileOutputStream("data.xlsx");
    workbook.write(fileOut);
    fileOut.close();
    workbook.close();
}
```

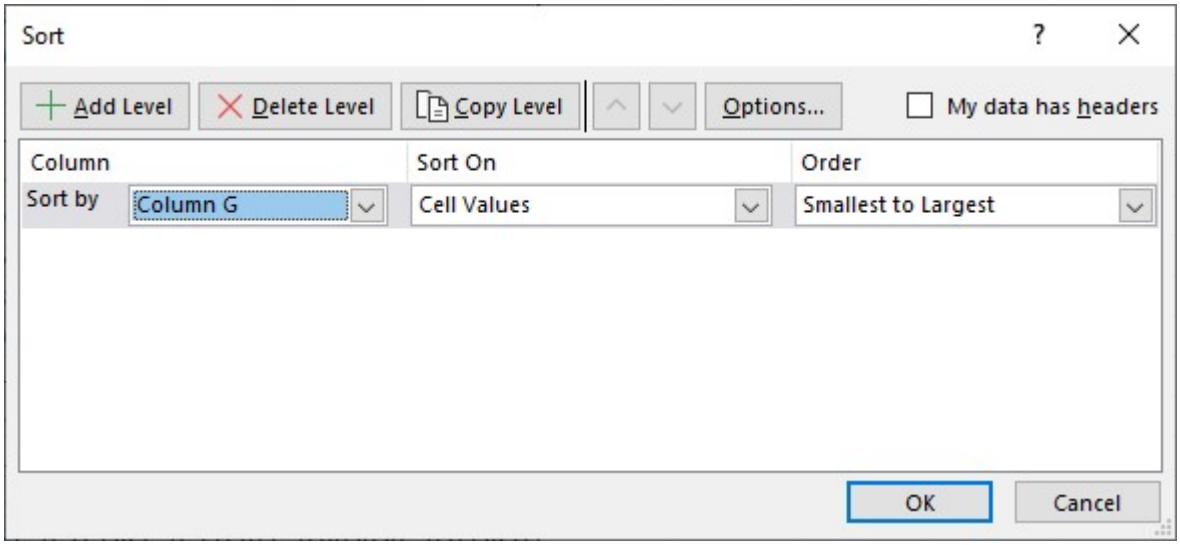
I could then upload the data into a new Excel file where I could then split the data manually into training, validation and testing data.

Data Splitting

Now I had all of the data cleaned and standardised into one file, I needed to split the rows by 60:20:20 with training, validation and testing data respectively

To prevent seasonal biases from occurring, I “shuffled” the data manually in excel, by assigning a random real number between 0 and 1 to each row, and ordered the entire table by the ordering of that column

	A	B	C	D	E	F	G	H
1	0.273272	0.308224	0.210108	0.623077	0.721176	0.179339	0.440205	
2	0.372811	0.324766	0.601083	0.5	0.702353	0.304959	0.543984	
3	0.560829	0.234112	0.369073	0.5	0.664706	0.252066	0.011289	
4	0.431797	0.363084	0.375812	0.5	0.787059	0.22562	0.581713	
5	0.487097	0.326916	0.662214	0.469231	0.74	0.35124	0.467049	
6	0.299078	0.381869	0.240794	0.469231	0.655294	0.199174	0.348347	
7	0.424424	0.372056	0.365584	0.346154	0.636471	0.304959	0.959612	
8	0.487097	0.583645	0.487004	0.407692	0.655294	0.410744	0.470264	
9	0.553456	0.280561	0.442238	0.561538	0.363529	0.43719	0.900404	
10	0.464977	0.29757	0.744043	0.5	0.382353	0.516529	0.906106	
11	0.498157	0.290748	0.240554	0.438462	0.815294	0.152893	0.60714	
12	0.350691	0.282243	0.347292	0.376923	0.542353	0.278512	0.118058	
13	0.199539	0.315888	0.52864	0.715385	0.495294	0.31157	0.150126	
14	0.568203	0.38271	0.708785	0.376923	0.664706	0.450413	0.751831	
15	0.413364	0.269439	0.394705	0.469231	0.683529	0.232231	0.376237	
16	0.512903	0.313738	0.494344	0.438462	0.608235	0.331405	0.287678	
17	0.288018	0.269907	0.369555	0.376923	0.570588	0.232231	0.635568	
18	0.579263	0.349065	0.826113	0.5	0.655294	0.516529	0.768822	
19	0.512903	0.373738	0.811432	0.284615	0.702353	0.463636	0.008863	
20	0.608756	0.405701	0.51769	0.315385	0.551765	0.463636	0.638314	
21	0.464977	0.329907	0.714922	0.407692	0.711765	0.377686	0.067392	
22	0.247465	0.233271	0.386402	0.684615	0.485882	0.258678	0.932818	
23	0.365438	0.266916	0.315644	0.561538	0.749412	0.166116	0.268815	
24	0.240092	0.758598	0.201083	0.407692	0.627059	0.338017	0.426953	
25	0.424424	0.340561	0.450782	0.5	0.598824	0.31157	0.339357	
26	0.350691	0.21243	0.363779	0.561538	0.485882	0.245455	0.675628	
27	0.405991	0.293738	0.223225	0.5	0.664706	0.219008	0.43897	
28	0.479724	0.450841	0.768592	0.5	0.551765	0.549587	0.599443	
29	0.46129	0.225607	0.337425	0.592308	0.184706	0.397521	0.832145	
30	0.678802	0.393364	0.777136	0.469231	0.655294	0.542975	0.781791	
31	0.394931	0.465327	0.748375	0.623077	0.58	0.483471	0.290774	
32	0.347005	0.630935	0.287726	0.469231	0.627059	0.318182	0.112709	
33	0.590323	0.262617	0.522503	0.530769	0.222353	0.536364	0.601746	
34	0.708295	0.438505	0.701444	0.346154	0.589412	0.576033	0.663562	
35	0.431797	0.308598	0.679543	0.561538	0.683529	0.357851	0.045043	
36	0.664055	0.423178	0.854633	0.376923	0.608235	0.615702	0.243355	
37	0.321198	0.350374	0.229122	0.530769	0.721176	0.18595	0.140906	


Sort
Add Level
Delete Level
Copy Level
Options...
My data has headers
Column
Sort On
Order
Sort by
Column G
Cell Values
Smallest to Largest
OK
Cancel

	A	B	C	D	E	F	G	H
1	0.52765	0.451682	0.77485	0.469231	0.664706	0.503306	0.646451	
2	0.487097	0.352056	0.412996	0.438462	0.768235	0.245455	0.450449	
3	0.542396	0.346916	0.382671	0.530769	0.702353	0.271901	0.718449	
4	0.48341	0.347757	0.376414	0.407692	0.796471	0.219008	0.514293	
5	0.512903	0.373738	0.811432	0.284615	0.702353	0.463636	0.539386	
6	0.512903	0.372056	0.410951	0.438462	0.721176	0.291736	0.872218	
7	0.597696	0.408692	0.480987	0.530769	0.664706	0.377686	0.831156	
8	0.376498	0.313738	0.388327	0.438462	0.344706	0.344628	0.158736	
9	0.560829	0.234112	0.369073	0.5	0.664706	0.252066	0.889135	
10	0.608756	0.354579	0.805535	0.376923	0.664706	0.509917	0.202332	
11	0.627189	0.377196	0.816606	0.376923	0.561176	0.602479	0.66552	
12	0.645622	0.412056	0.737906	0.346154	0.627059	0.516529	0.812758	
13	0.708295	0.312897	0.618412	0.530769	0.128235	0.734711	0.674484	
14	0.594009	0.391215	0.65716	0.438462	0.645882	0.463636	0.238911	
15	0.571889	0.423178	0.670878	0.469231	0.683529	0.423967	0.882792	
16	0.276959	0.180935	0.342118	0.807692	0.495294	0.22562	0.325464	
17	0.54977	0.372897	0.73911	0.469231	0.589412	0.496694	0.930083	
18	0.53871	0.369065	0.801805	0.407692	0.692941	0.470248	0.207186	
19	0.48341	0.356729	0.659567	0.438462	0.655294	0.404132	0.90621	
20	0.358065	0.211963	0.372202	0.469231	0.344706	0.304959	0.793846	
21	0.361751	0.245234	0.431769	0.684615	0.627059	0.238843	0.973931	
22	0.52765	0.297103	0.457882	0.376923	0.250588	0.443802	0.371867	
23	0.372811	0.216636	0.491336	0.623077	0.401176	0.338017	0.133464	
24	0.546083	0.422336	0.819134	0.5	0.627059	0.536364	0.828091	
25	0.435484	0.375047	0.45006	0.530769	0.711765	0.285124	0.377166	
26	0.339631	0.364019	0.512996	0.561538	0.278824	0.43719	0.546658	
27	0.417051	0.34271	0.5787	0.346154	0.627059	0.390909	0.136474	
28	0.210599	0.675234	0.377617	0.376923	0.156471	0.569421	0.538538	
29	0.457604	0.45	0.663658	0.469231	0.664706	0.404132	0.570797	
30	0.50553	0.255794	0.658724	0.5	0.476471	0.404132	0.255334	
31	0.616129	0.392056	0.615644	0.469231	0.721176	0.390909	0.562719	
32	0.601382	0.45215	0.763538	0.438462	0.655294	0.52314	0.844418	
33	0.332258	0.275421	0.208063	0.469231	0.692941	0.179339	0.699484	
34	0.501843	0.426542	0.350903	0.438462	0.777647	0.238843	0.099646	
35	0.424424	0.216636	0.327677	0.592308	0.335294	0.331405	0.181423	
36	0.612442	0.36271	0.793622	0.376923	0.674118	0.503306	0.611124	
37	0.457604	0.275421	0.285921	0.592308	0.749412	0.179339	0.905605	

I could then simply delete the last column and perform the 60:20:20 split into each of the respective files and dragged them into the root of my BackPropagation file.

Implementation of MLP

For my implementation, I have decided to code it in java as the object oriented nature goes hand in hand with the different approaches of training the models, where each of the different training models can be represented as objects with slight alterations to the original model, where the altered functionality can be handled using inheritance and polymorphism.

Basic Model

```
public class Basic {
    protected final ArrayList<ArrayList<Double>> trainingData = BackPropagation.trainingData;
    protected final ArrayList<ArrayList<Double>> validationData = BackPropagation.validationData;
    protected double[][] weights;
    protected double[] biases;
    protected int noInput;
    protected int noHidden;
    protected int noEpochs;
    protected double p; //Learning Parameter
    protected double mse;

    public Basic(int noInput, int noHidden, int noEpochs, double p){
        this.weights=new double[noInput+1][noHidden];
        this.biases=new double[noHidden+1];
        this.noInput=noInput;
        this.noHidden=noHidden;
        this.noEpochs=noEpochs;
        this.p=p;
        this.mse=0;
        this.createNetwork();
    }
```

The original MLP model is stored inside the “Basic” class, which has all the basic parameters needed for an MLP, stored and instantiated in its constructor.

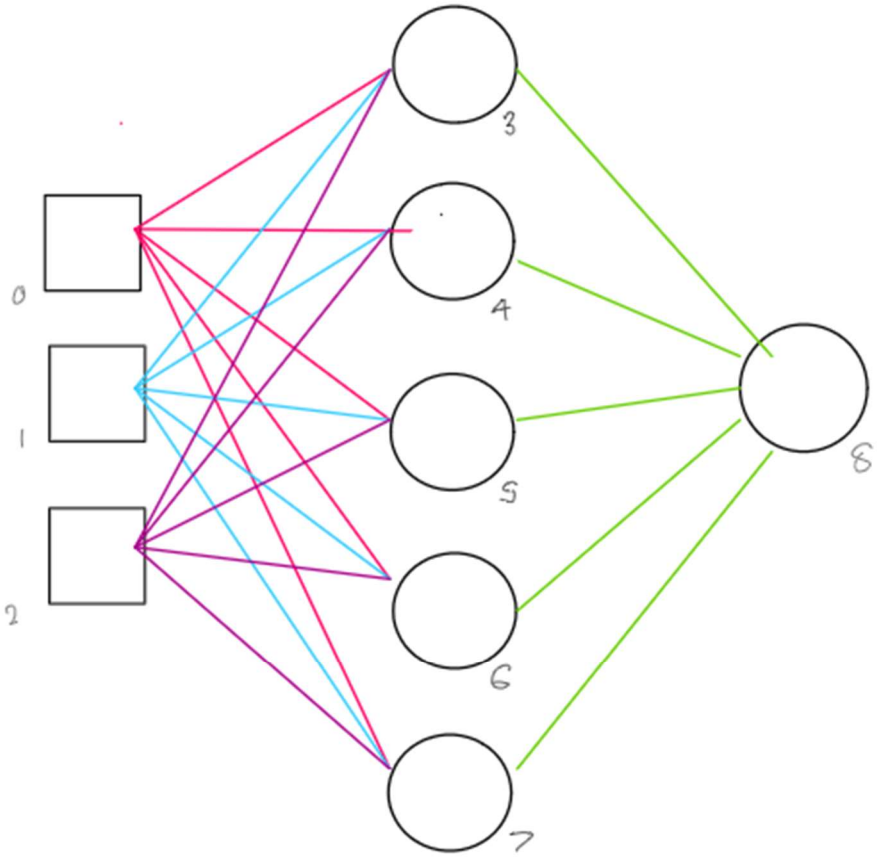
```
Basic b = new Basic(5,4,1000,0.1);
```

Inside the main class, you can instantiate the object with any number of inputs (up to 5 in this case), any number of hidden nodes, one output node, and any number of epochs.

```
public void createNetwork(){
    //create the weights from the input to hidden nodes
    Random rnd = new Random();
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            this.weights[i][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
        }
    }

    //create the weights from hidden to output node
    for (int j = 0; j < noHidden; j++) {
        this.weights[noInput][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
    }

    //create the biases for the hiddens and output node
    for (int i = 0; i < noHidden + 1; i++) {
        this.biases[i] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput); //random value between [-2/n,2/n]
    }
}
```

Once the object is created, it will immediately initialize the weights and biases to random values between [-2/n,2/n]

The dimensions of the weights and biases will be based on the number of inputs and hidden nodes you decided to use when instantiating the object



$$\text{weights} = [[w_{03}, w_{04}, w_{05}, w_{06}, w_{07}],$$
$$[w_{13}, w_{14}, w_{15}, w_{16}, w_{17}],$$
$$[w_{23}, w_{24}, w_{25}, w_{26}, w_{27}],$$
$$[w_{38}, w_{48}, w_{58}, w_{68}, w_{78}]]$$

$$\text{biases} = [b_{U_3}, b_{U_4}, b_{U_5}, b_{U_6}, b_{U_7}, b_{U_8}]$$

$$U = [U_0, U_1, U_2, U_3, U_4, U_5, U_6, U_7, U_8]$$

The above figure shows how my weights, biases and u values are stored on my program

```
public void run(){
    for (int i = 0; i < noEpochs+1; i++) {
        if(i % 500 != 0){ //Training cycle
            training();
            //printWeights(weights,biases);
        }else{ //Validation cycle
            validation();
            plotTrainingVsTesting(i);
        }
    }
    testing();
    printWeights();
}
```

```
public void training(){
    for (int j = 0; j < trainingData.size(); j++) {
        double[] u = new double[noInput + noHidden + 1]; //create input set
        double[] deltas = new double[noHidden + 1];
        for (int l = 0; l < noInput; l++) {
            u[l]=trainingData.get(j).get(l); //stores all the starting values in u list
        }
        double  output=trainingData.get(j).get(trainingData.get(j).size()-1); //gets the output value at the end of the list
        forwardPass(u);
        backwardPass(u,deltas,output);
    }
}
```

```
public void validation(){
    double[] predictedOutputs = new double[validationData.size()];
    double[] actualOutputs = new double[validationData.size()];
    //initialize u inputs and outputs
    for (int j = 0; j < validationData.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = validationData.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]=deStandardiseOutput(validationData.get(j).get(validationData.get(j).size()-1));
    }
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < validationData.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    mse = total/validationData.size(); //update MSE
    System.out.println("Validation MSE: " + mse);
}
```

```
public void testing(){
    double[] predictedOutputs = new double[testingData.size()];
    double[] actualOutputs = new double[testingData.size()];
    for (int j = 0; j < testingData.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = testingData.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]=deStandardiseOutput(testingData.get(j).get(testingData.get(j).size()-1));
    }
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < testingData.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    mse = total/testingData.size(); //update MSE
    //System.out.println("MSE: " + mse);
    System.out.println("Testing MSE: " +(total/testingData.size()));
    plotPredVsActual(testingData,predictedOutputs,actualOutputs);
}
```

```
public double deStandardiseOutput(double value){
    return (((value-0.1)/0.8)*(1.28-0.07)+0.07);
}
```

I've made the loop go up to noEpochs+1 just so it can have an additional validation cycle at the beginning and end

The algorithm works by first doing a validation cycle to get the first MSE, followed by 500 epochs of training, 1 cycle of validation, and so on, until the number of epochs is met.

```
public void training(){
    for (int j = 0; j < trainingData.size(); j++) {
        double[] u = new double[noInput + noHidden + 1]; //create input set
        double[] deltas = new double[noHidden + 1];
        for (int l = 0; l < noInput; l++) {
            u[l]=trainingData.get(j).get(l); //stores all the starting values in u list
        }
        double  output=trainingData.get(j).get(trainingData.get(j).size()-1); //gets the output value at the end of the list
        forwardPass(u);
        backwardPass(u,deltas,output);
    }
}
```

The training cycle first creates a new array u, which will store the values of each of the nodes. We can populate the first noInput nodes by grabbing them from the trainingData row, and set the expected output to be the final index of the trainingData row.

We can then run the forward pass on the u array, which will generate all the consequent u values in the hidden and output layer

```
public double forwardPass(double[] u){
    //forward pass for the weights from the input to hidden nodes
    double S = 0;
    for(int i=0; i<noHidden; i++){
        S=biases[i];
        for (int j = 0; j < noInput ; j++) {
            S += weights[j][i]*u[j];
        }
        u[i+noInput] = sigmoid(S);
    }

    //pass from the hidden nodes to the output node
    S=biases[biases.length-1];
    for (int i = 0; i < noHidden; i++) {
        S += weights[weights.length-1][i]*u[noInput+i];
    }
    u[u.length-1] = sigmoid(S);

    return u[u.length-1]; //returns the final predicted output for that row
}
```

In the forward pass function, we have one for loop to go through each weight from the input to hidden nodes, get the weighted sum of each of the input to hidden nodes, for that input node, and set the u value to the sigmoid of the weighted sum.

In the next for loop, we apply the sigmoid function to the weighted sum of the weights between the output and hidden nodes, and set it to the final u value.

```
public void backwardPass(double[] u, double[] deltas, double output){
    double deriv = u[u.length-1]*(1-u[u.length-1]); //gets the derivative of the final output where u[noInput+noHidden] is the last element of u
    deltas[deltas.length-1] = (output - u[u.length-1])*deriv; //get last delta value
    //Construct the rest of the deltas using this
    for (int i = 0; i < noHidden; i++) {
        double newDeriv = u[noInput+i]*(1-u[noInput+i]);
        deltas[i] = weights[weights.length-1][i]*deltas[deltas.length-1]*newDeriv;
    }
    //update weights from input to hidden
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            weights[i][j] += (p*deltas[j]*u[i]);
        }
    }
    //update biases for hidden layer
    for (int i = 0; i < noHidden; i++) {
        biases[i] += (p*deltas[i]);
    }

    //update weights from hidden to output
    for (int i = 0; i < noHidden; i++) {
        weights[weights.length-1][i] += (p*deltas[deltas.length-1]*u[i+noInput]);
    }

    //update bias for output node
    biases[biases.length-1] += (p*deltas[deltas.length-1]);
}
```

We can then use these calculated u values to adjust the weights and biases

We do this by first getting the derivative of the output of the sigmoid function, which is stored in the final u value, which can be calculated using a simple calculation:

$$f'(S_j) = u_j(1-u_j)$$

We can then calculate the delta value for the final node, which will be

$$\delta_O = (C-u_O)\, f'(S_O)$$

Where C is the expected output, u_0 is the predicted output (stored as the final u value for our solution) multiplied by the derivative of the output of the sigmoid function, which we set above.

Now we have these two base values set, we can generate the delta value for the hidden nodes using this formula:

$$\delta_j = w_{j,O}\delta_O f'(S_j)$$

Where $w_{j,0}$ is the weight from the hidden node j to the output node 0. And $f'(S_j)$ is the derivative of the sigmoid of hidden node j.

```
for (int i = 0; i < noHidden; i++) {
    double newDeriv = u[noInput+i]*(1-u[noInput+i]);
    deltas[i] = weights[weights.length-1][i]*deltas[deltas.length-1]*newDeriv;
}
```

Since these deltas can be done in any order, we start from 0 to the number of hidden nodes in the network. And since our hidden-to-output weights are stored in the bottom row, we can access these using row index “weights.length-1” and column index “i”.

Now we have all the delta values, we can update the weights using the formula:

$$w_{i,j}^{*} = w_{i,j} + \rho\delta_j u_i$$

```
//update weights from input to hidden
for (int i = 0; i < noInput; i++) {
    for (int j = 0; j < noHidden; j++) {
        weights[i][j] += (p*deltas[j]*u[i]);
    }
}

//update weights from hidden to output
for (int i = 0; i < noHidden; i++) {
    weights[weights.length-1][i] += (p*deltas[deltas.length-1]*u[i+noInput]);
}
```

Then we update the biases using this formula:

$$b_j^{*} = b_j + \rho\delta_j$$

Note: It doesn't include a u_i since biases only have one node associated with it, not two.

```
//update biases for hidden layer
for (int i = 0; i < noHidden; i++) {
    biases[i] += (p*deltas[i]);
}

//update bias for output node
biases[biases.length-1] += (p*deltas[deltas.length-1]);
```

```
public void validation(){
    double[] predictedOutputs = new double[validationData.size()];
    double[] actualOutputs = new double[validationData.size()];
    //initialize u inputs and outputs
    for (int j = 0; j < validationData.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = validationData.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]=deStandardiseOutput(validationData.get(j).get(validationData.get(j).size()-1));
    }
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < validationData.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    mse = total/validationData.size(); //update MSE
    System.out.println("Validation MSE: " + mse);
}
```

The validation cycle for the Basic model simply just gets the Mean Squared error every 500 epochs using the validation data set, so the user can figure out the optimal number of epochs before overtraining starts to occur

It first initializes two arrays predictedOutputs and actualOutputs which store the predicted output from the forward pass vs. the actual output from the dataset respectively.

It will then go through a for loop through each row of inputs and outputs, initialize the u values for each of the input values, and then returns the predicted output from the forwardPass

```
public double forwardPass(double[] u){
    //forward pass for the weights from the input to hidden nodes
    double S = 0;
    for(int i=0; i<noHidden; i++){
        S=biases[i];
        for (int j = 0; j < noInput ; j++) {
            S += weights[j][i]*u[j];
        }
        u[i+noInput] = sigmoid(S);
    }

    //pass from the hidden nodes to the output node
    S=biases[biases.length-1];
    for (int i = 0; i < noHidden; i++) {
        S += weights[weights.length-1][i]*u[noInput+i];
    }
    u[u.length-1] = sigmoid(S);

    return u[u.length-1]; //returns the final predicted output for that row
}
```

This is why I made the forwardPass function return the final u value (which would be the predictand) as it's a bit more direct.

It then gets the output value for that row and puts it in the actualOutputs array

It does this for each row in the validation data

It now has the resources to perform the MSE calculation

```
double total = 0;
//Mean Squared Error calculation
for (int j = 0; j < validationData.size(); j++) {
    total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
}
mse = total/validationData.size(); //update MSE
System.out.println("MSE: " + mse);
```

It just adds the difference between the predicted and output values squared, to a total, and divides it by the number of rows

$$MSE = \frac{\sum(O - M)^2}{n}$$

Improvements

For the improvements, I decided to use inheritance to extend the functionality of the Basic Model, as each of the improvement shared the same parameters and functionality of the Basic Model, with additional parameters and alterations to some functionality, which could be done via polymorphism

Momentum

The first improvement I tried making was momentum as I felt this was the most simple and intuitive improvement

```
package backpropagation;

import java.util.Random;

public class Momentum extends Basic{

    private double[][] prevWeights;
    private double[] prevBiases;
    private double a;

    public Momentum(int noInput, int noHidden, int noEpochs, double p, double a){
        super(noInput,noHidden,noEpochs,p);
        this.a=a;
    }


    @Override
    public void createNetwork(){
        this.prevWeights=new double[noInput+1][noHidden];
        this.prevBiases=new double[noHidden+1];
        //create the weights from the input to hidden nodes
        Random rnd = new Random();
        for (int i = 0; i < noInput; i++) {
            for (int j = 0; j < noHidden; j++) {
                this.weights[i][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
                this.prevWeights[i][j] = weights[i][j];
            }
        }

        //create the weights from hidden to output node
        for (int j = 0; j < noHidden; j++) {
            this.weights[noInput][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
            this.prevWeights[noInput][j] = weights[noInput][j];
        }

        //create the biases for the hiddens and output node
        for (int i = 0; i < noHidden + 1; i++) {
            this.biases[i] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput); //random value between [-2/n,2/n]
            this.prevBiases[i]=biases[i];
        }

    }
```

Momentum has 3 additional fields:

- prevWeights and prevBiases – To store the previous weight changes and get the weight change so the algorithm can keep “pushing” in that direction if a significant enough improvement was made
- alpha (a) – This is the momentum coefficient that determines how much additional “push” will occur to the weight modification, based on the weight change

We initialize the momentum coefficient in the constructor, and override the network construction to also set the prevWeights to what the current weights are

```java
@Override
public void backwardPass(double[] u, double[] deltas, double output){
    double deriv = u[u.length-1]*(1-u[u.length-1]); //gets the derivative of the final output where u[noInput+noHidden] is the last element of u
    deltas[deltas.length-1] = (output - u[u.length-1])*deriv; //get last delta value

    double change;
    //Construct the rest of the deltas using this
    for (int i = 0; i < noHidden; i++) {
        double newDeriv = u[noInput+i]*(1-u[noInput+i]);
        deltas[i] = weights[noInput][i]*deltas[deltas.length-1]*newDeriv;
    }
    //update weights from input to hidden
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            change = weights[i][j] - prevWeights[i][j];
            prevWeights[i][j] = weights[i][j];
            weights[i][j] += (p*deltas[j]*u[i]) + (a*change);
        }
    }
    //update biases for hidden layer
    for (int i = 0; i < noHidden; i++) {
        change = biases[i] - prevBiases[i];
        prevBiases[i] = biases[i];
        biases[i] += (p*deltas[i]) + (a*change);
    }

    //update weights from hidden to output
    for (int i = 0; i < noHidden; i++) {
        change = weights[noInput][i] - prevWeights[noInput][i];
        prevWeights[noInput][i] = weights[noInput][i];
        weights[noInput][i] += (p*deltas[deltas.length-1]*u[i+noInput]) + (a*change);
        //should be noInput I think?
    }

    //update bias for output node
    change = biases[biases.length-1] - prevBiases[prevBiases.length-1];
    prevBiases[prevBiases.length-1] = biases[biases.length-1];
    biases[biases.length-1] += (p*deltas[deltas.length-1]) + (a*change);
}
```

We also override the backwardPass function

Only additional change we make to the functionality is we modify how the weights get updated

We use the following formula(s):

$$\Delta w_{i,j} = w^{*}_{i,j} - w_{i,j}$$

We calculate the change in weight by subtracting the previous weight from the current

```java
change = weights[i][j] - prevWeights[i][j];
```

$$w^{*}_{i,j} = w_{i,j} + \rho\delta_j u_i + \alpha\Delta w_{i,j}$$

And add the momentum onto the weight modification

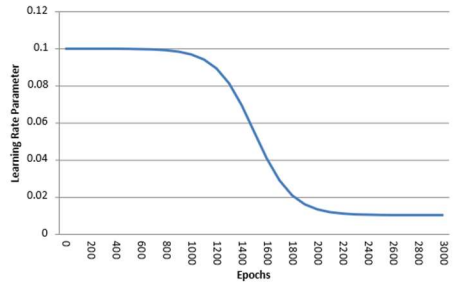
```java
weights[i][j] += (p*deltas[j]*u[i]) + (a*change);
```

Annealing

```java
public class Annealing extends Basic{

    private double ep;

    public Annealing(int noInput, int noHidden, int noEpochs, double p, double ep) {
        super(noInput, noHidden, noEpochs, p);
        this.ep=ep;
    }
```

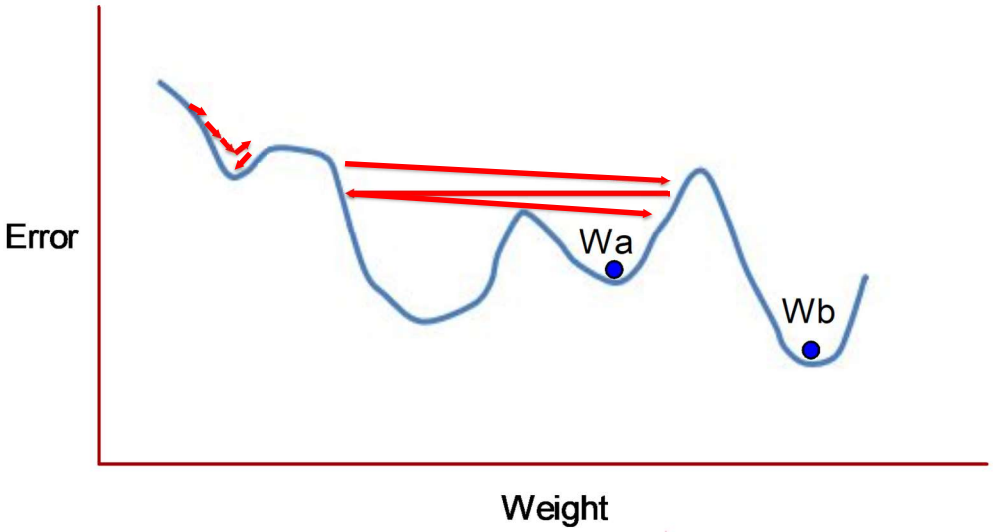
We have an additional parameter ep (end learning parameter). And now our “p” from the Basic model is interpreted as the start parameter

$$f(x) = p + (q - p)\left(1 - \frac{1}{1 + e^{10 - \frac{20x}{r}}}\right)$$



Example:
p = end parm: 0.01
q = start parm: 0.1
r = max epochs = 3000
x = epochs so far

The idea behind it is simillar to bold driver in the sense of having a high learning parameter at the start as the algorithm is continuously oscillating around each of the local minima’s in order to find the global minima, and once the global minima is found later in the epochs, the learning parameter is then decreased to prevent potential “jumping” over to a new minima.




```
public double p(int x){
    return ep+(p-ep)*(1-(1/(1+Math.exp(10-((20*(x))/noEpochs))))); //returns learning parameter based on current number of epochs
}
```

We create a new function that returns the current learning parameter value based on the start parameter (p), end parameter (ep), no of epochs so far (x) and max epochs (noEpochs)

```
@Override
public void train(){
double[] u;
double[] deltas;
double output;

for (int i = 0; i < noEpochs+1; i++) {
    if(i % 500 != 0){ //Training cycle
        for (int j = 0; j < trainingData.size(); j++) {
            u = new double[noInput + noHidden + 1]; //create input set
            deltas = new double[noHidden + 1];
            for (int l = 0; l < noInput; l++) {
                u[l]=trainingData.get(j).get(l); //stores all the starting values in u list
            }
            output=trainingData.get(j).get(trainingData.get(j).size()-1); //gets the output value at the end of the list
            forwardPass(u);
            backwardPass(u,deltas,output,i);
        }
        //printWeights(weights,biases);
```

```
public void backwardPass(double[] u, double[] deltas, double output, int x){
    double deriv = u[u.length-1]*(1-u[u.length-1]); //gets the derivative of the final output where u[noInput+noHidden] is the last element of u
    deltas[deltas.length-1] = (output - u[u.length-1])*deriv; //get last delta value
    //Construct the rest of the deltas using this
    for (int i = 0; i < noHidden; i++) {
        double newDeriv = u[noInput+i]*(1-u[noInput+i]);
        deltas[i] = weights[noInput][i]*deltas[deltas.length-1]*newDeriv;
    }
    //update weights from input to hidden
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            weights[i][j] += (p(x)*deltas[j]*u[i]);
        }
    }
    //update biases for hidden layer
    for (int i = 0; i < noHidden; i++) {
        biases[i] += (p(x)*deltas[i]);
    }

    //update weights from hidden to output
    for (int i = 0; i < noHidden; i++) {
        weights[noInput][i] += (p(x)*deltas[deltas.length-1]*u[i+noInput]);
    }

    //update bias for output node
    biases[biases.length-1] += (p(x)*deltas[deltas.length-1]);
}
```

We override the backward pass so instead of inputting “p” when modifying the weights, we input the function along with the number of epochs executed so far

Bold Driver

```
public class BoldDriver extends Basic{

    private LearningParameter p;
    private double[][] weightChanges;
    private double[] biasChanges;
    private Double prevMse;

    public BoldDriver(int noInput, int noHidden, int noEpochs, LearningParameter p) {
        super(noInput, noHidden, noEpochs);
        this.p=p;
        this.prevMse=null;
    }
```

```java
@Override
public void createNetwork(){
    this.weightChanges = new double[noInput+1][noHidden];
    this.biasChanges = new double[noHidden+1];
    //create the weights from the input to hidden nodes
    Random rnd = new Random();
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            this.weights[i][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
        }
    }

    //create the weights from hidden to output node
    for (int j = 0; j < noHidden; j++) {
        this.weights[noInput][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
    }

    //create the biases for the hiddens and output node
    for (int i = 0; i < noHidden + 1; i++) {
        this.biases[i] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput); //random value between [-2/n,2/n]
    }
}
```

For this example, instead of having just a learning parameter p as a double, we create an object learning parameter where the value can be adjusted depending on how much the MSE changes in each cycle

We also initialize weight and bias changes variables to be the same size as the weights and biases arrays respectively

```java
public class LearningParameter {
    private double value;
    private double min;
    private double max;
    private double inc;
    private double dec;

    public LearningParameter(double value,double min, double max, double inc,double dec){
        this.value=value;
        this.min=min;
        this.max=max;
        this.inc=1+inc;
        this.dec=1-dec;
    }

    public double getValue(){
        return value;
    }

    public void increase(){
        if((this.value*inc)>=max){
            this.value=max;
        }else{
            this.value*=inc;
        }
    }

    public void decrease(){
        if((this.value*dec)<=min){
            this.value=min;
        }else{
            this.value*=dec;
        }
    }
```

We still have the value, which will follow the same principle as the ordinary learning parameter, but now we can modify the learning parameter by increasing/decreasing it by a certain percentage, without it exceeding the min/max respectively

```
@Override
public void backwardPass(double[] u, double[] deltas, double output){
    double deriv = u[u.length-1]*(1-u[u.length-1]); //gets the derivative of the final output where u[noInput+noHidden] is the last element of u
    deltas[deltas.length-1] = (output - u[u.length-1])*deriv; //get last delta value
    //Construct the rest of the deltas using this
    for (int i = 0; i < noHidden; i++) {
        double newDeriv = u[noInput+i]*(1-u[noInput+i]);
        deltas[i] = weights[noInput][i]*deltas[deltas.length-1]*newDeriv;
    }
    //update weights from input to hidden
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            weights[i][j] += (p.getValue()*deltas[j]*u[i]);
            weightChanges[i][j] = (p.getValue()*deltas[j]*u[i]); //update the weight changes
        }
    }
    //update biases for hidden layer
    for (int i = 0; i < noHidden; i++) {
        biases[i] += (p.getValue()*deltas[i]);
        biasChanges[i] = (p.getValue()*deltas[i]); //update the bias changes
    }

    //update weights from hidden to output
    for (int i = 0; i < noHidden; i++) {
        weights[noInput][i] += (p.getValue()*deltas[deltas.length-1]*u[i+noInput]);
        weightChanges[noInput][i] = (p.getValue()*deltas[deltas.length-1]*u[i+noInput]);
    }

    //update bias for output node
    biases[biases.length-1] += (p.getValue()*deltas[deltas.length-1]);
    biasChanges[biasChanges.length-1] = (p.getValue()*deltas[deltas.length-1]);
}
```

We Override the backwardPass function to use p.getValue() instead of p to return the actual value instead of the object itself

We also update the weight and bias changes by their respective amounts from their calculations each time the weights are updated

```
@Override
public void validation(){
    super.validation();
    if(prevMse != null){
        double percentChange = ((mse-prevMse)/prevMse)*100;
        if(percentChange>=0.02){ //error function increase
            System.out.println("inc");
            revert(); //reverts the weight and bias changes
            p.decrease();
        }else if(percentChange<=0){ //error function decrease
            System.out.println("dec");
            p.increase();
        }
    }
    prevMse = mse;
}
```

In addition to the validation cycle, we store the previous MSE from the previous validation cycle, so we can get the percentage difference of the MSE between the current and previous validation cycle.

In this case, if the percentage difference increase by 0.1% or more, it could mean that the learning rate was too large and could potentially be oscillating to another local minima. So the algorithm reverts the weight changes and decreases the weight by x% (x is determined by the learning parameter object) in an attempt to stabilize the weight adjustment. And If the error function decreases by any amount, it means its going in the correct direction, but perhaps too slowly, so the algorithm increases the learning parameter by y% (y is determined by the learning parameter object).

```
public void revert(){
    //set weights to prevWeights
    for (int i = 0; i < weights.length-1; i++) {
        for (int j = 0; j < weights[0].length; j++) {
            weights[i][j]-=weightChanges[i][j];
        }
    }
    //set biases to prevBiases
    for (int i = 0; i < biases.length-1; i++) {
        biases[i]-=biasChanges[i];
    }
}
```

The weights and biases can be easily reverted by using for loops and decrementing each index by its respective previous weight change.

I include the Bold Driver adjustments inside the Validation cycle, as the validation cycle occurs once every 500 epochs, so the Bold Driver only updates every 500 epochs instead of every epoch, which prevents the adjustments from being too severe.

Also, as stated before, I've added min and max as "safety nets" for the LearningParameter object to prevent it from either oscillating too much, or too little.

Weight Decay

```
public class WeightDecay extends Basic{

    public WeightDecay(int noInput, int noHidden, int noEpochs, double p) {
        super(noInput, noHidden, noEpochs, p);
    }
```

There are no additional field variables as the only modifications it makes are based off of the current weight values, and current number of epochs, which we already both have from our Basic model

```
@Override
public void run(){
    for (int i = 0; i < noEpochs+1; i++) {
        if(i % 500 != 0){ //Training cycle
            training(i);
        }else{ //Validation cycle
            validation();
            plotTrainingVsTesting(i);
        }
    }
    testing();
    printWeights();
}
```

```
public void training(int i){
    for (int j = 0; j < trainingData.size(); j++) {
        double[] u = new double[noInput + noHidden + 1]; //create input set
        double[] deltas = new double[noHidden + 1];
        for (int l = 0; l < noInput; l++) {
            u[l]=trainingData.get(j).get(l); //stores all the starting values in u list
        }
        double output=trainingData.get(j).get(trainingData.get(j).size()-1); //gets the output value at the end of the list
        forwardPass(u);
        backwardPass(u,deltas,output,i);
    }
}
```

We also need to make sure to override the run function, to instead call the training function with the number of epochs added. So we can add this modification to the backwardPass function:

```
public void backwardPass(double[] u, double[] deltas, double output, int x){
    double deriv = u[u.length-1]*(1-u[u.length-1]); //gets the derivative of the final output where u[noInput+noHidden] is the last element of u
    deltas[deltas.length-1] = (output - u[u.length-1] + (upsilon(x)*omega()))*deriv; //get last delta value
    //Construct the rest of the deltas using this
    for (int i = 0; i < noHidden; i++) {
        double newDeriv = u[noInput+i]*(1-u[noInput+i]);
        deltas[i] = weights[noInput][i]*deltas[deltas.length-1]*newDeriv;
    }
    //update weights from input to hidden
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            weights[i][j] += (p*deltas[j]*u[i]);
        }
    }
    //update biases for hidden layer
    for (int i = 0; i < noHidden; i++) {
        biases[i] += (p*deltas[i]);
    }

    //update weights from hidden to output
    for (int i = 0; i < noHidden; i++) {
        weights[noInput][i] += (p*deltas[deltas.length-1]*u[i+noInput]);
    }

    //update bias for output node
    biases[biases.length-1] += (p*deltas[deltas.length-1]);
}
```

We create a new backpropagation method that adds a penalty term to the error function

$$\delta_O = \underbrace{(C-u_O)}_{E} f'(S_O)$$

Becomes:

$$\delta_O = (C-u_O + \upsilon\Omega) f'(S_O)$$

```
public double omega(){
    double o = 0;
    for (int i = 0; i < weights.length; i++) {
        for (int j = 0; j < weights[0].length; j++) {
            o+=Math.pow(weights[i][j],2);
        }
    }
    for (int i = 0; i < biases.length; i++) {
        o+=Math.pow(biases[i],2);
    }
    o/=(2*((weights.length*weights[0].length)+biases.length));
    return o;
}
```

$$\Omega = \frac{1}{2n}\sum_{i=1}^{n} w_i^2$$

```
public double upsilon(int x){
    return 1/(p*x);
}
```

$$\upsilon = \frac{1}{\rho e}$$

Training and Network Selection

During this section, I will investigate, using the Basic BackPropagation algorithm:

- Whether I get better data with 5 inputs or, 4 inputs (without DSP)
- The number of epochs that gives us the minimum MSE for the testing dataset

5 Inputs

I will first investigate which number of hidden nodes gives the best MSE for 1000 epochs.

And then, using those number of hidden nodes, I will investigate which number of epochs will give the highest mean squared error value for the testing data set (i.e. Whats the highest number of epochs I can reach before overtraining starts to occur).

Hidden Nodes

I will investigate which number of hidden nodes is the most optimal by running each parameter for 1000 and 5000 epochs and plotting the Training MSE and Testing MSE, to both identify which one adjusts the weights most quickly (Training MSE), but also which one adapts the best to unseen data (Testing MSE).

```
@Override
public void run(){
    for (int i = 0; i < noEpochs+1; i++) {
        if(i % 500 != 0){ //Training cycle
            training(i);
        }else{ //Validation cycle
            validation();
            plotTrainingVsTesting(i);
        }
    }
    testing();
    printWeights();
}
```

```
public void testing(){
    double[] predictedOutputs = new double[testingData.size()];
    double[] actualOutputs = new double[testingData.size()];
    for (int j = 0; j < testingData.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = testingData.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]=deStandardiseOutput(testingData.get(j).get(testingData.get(j).size()-1));
    }
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < testingData.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    mse = total/testingData.size(); //update MSE
    //System.out.println("MSE: " + mse);
    System.out.println("Testing MSE: " +(total/testingData.size()));
    plotPredVsActual(testingData,predictedOutputs,actualOutputs);
}
```

```
public double getMSE(ArrayList<ArrayList<Double>> data){
    double[] predictedOutputs = new double[data.size()];
    double[] actualOutputs = new double[data.size()];
    //initialize u inputs and outputs
    for (int j = 0; j < data.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = data.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]= deStandardiseOutput(data.get(j).get((data.get(j).size())-1));
    }
    //printArray(actualOutputs);
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < data.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    return (total/data.size());
}
```

```
public double deStandardiseOutput(double value){
    return (((value-0.1)/0.8)*(1.28-0.07)+0.07);
}
```

When getting the predicted and actual outputs, I make sure to de-standardise the data using the max and min values from the output dataset

1000 Epochs

No of Hidden Nodes	Training MSE	Testing MSE
2	6.332795527102071E-4	9.043843321208301E-4
3	5.976893302616521E-4	8.621352647486133E-4
4	6.010146353414403E-4	8.742132478142974E-4
5	6.112718158665903E-4	8.809489559266076E-4
6	6.064854458466744E-4	8.674957071682863E-4
7	6.303836233723303E-4	8.854915292115902E-4
8	6.068496774460654E-4	8.640672224443291E-4
9	6.502860331278691E-4	9.037204315498702E-4
10	6.088544074852629E-4	8.707208345413974E-4

5000 epochs

No of Hidden Nodes	Training MSE	Testing MSE
2	5.696637589669022E-4	8.496106201779735E-4
3	5.507949640016919E-4	8.241718441453457E-4
4	5.487110579727555E-4	8.181169728092502E-4
5	5.551019109761504E-4	8.25946117278259E-4
6	5.569140665511057E-4	8.325617989539788E-4
7	5.67641588311495E-4	8.343151666133807E-4
8	5.596584893025301E-4	8.316324805648771E-4
9	5.661981150661746E-4	8.250661354633327E-4
10	5.524097174522978E-4	8.126635741606624E-4

From this data, I can conclude that the most optimal no of hidden nodes is between 3, 4 and 10.

It's very difficult to tell which one is more optimal, so I will plot each set of MSE's against a graph over the number of epochs, to investigate which one has the lowest Testing MSE minima (i.e. which one will train the data more accurately) and how many Epochs it takes to reach that point

No of Epochs

Since we have 3 different modifications, and I want to be using the same starting weights, but also making them random at the same time, I will use a “seed value” for the Random object, to ensure that the random values are still random in between the correct ranges of [-2/n,2/n], but also will always return the exact same values

3 Hidden Nodes

I will now investigate the maximum number of epochs 3 hidden nodes will take before overtraining starts to occur, and will use this to identify the optimal number of epochs to produce the most accurate data

To do this, I will need to plot the no of epochs against the error rate in the form of a graph. So I added additional functionality to my code:

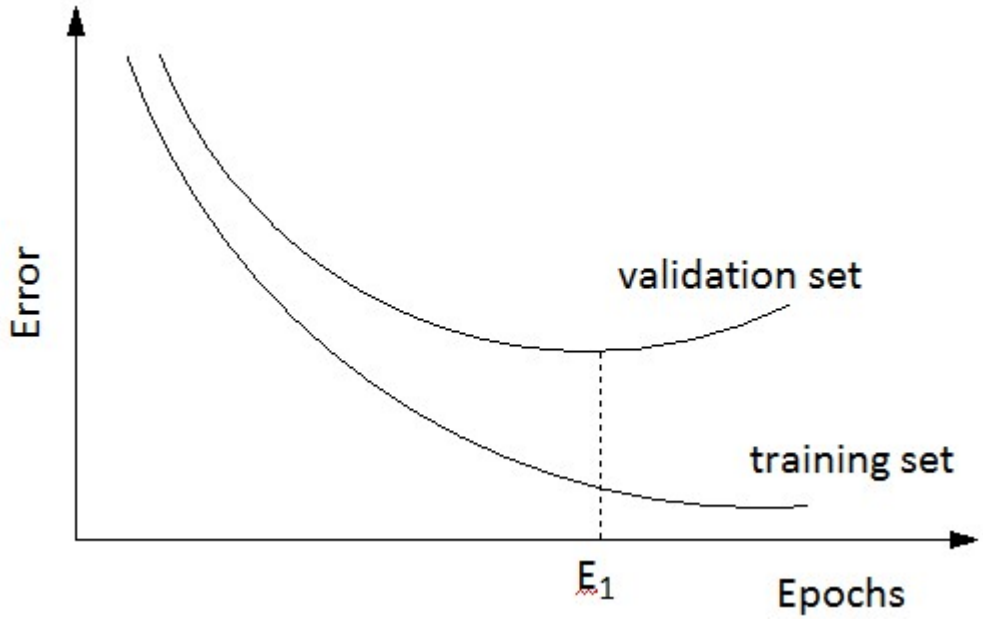
```
public void run(){
    for (int i = 0; i < noEpochs+1; i++) {
        if(i % 500 != 0){ //Training cycle
            training();
            //printWeights(weights,biases);
        }else{ //Validation cycle
            //validation();
            plotTrainingVsTesting(i);
        }
    }
}
```

```
public void plotTrainingVsTesting(int i){
    double trainingMSE = getMSE(trainingData);
    double testingMSE = getMSE(testingData);
    try {
        FileInputStream fis = new FileInputStream(new File("ErrorPlot.xlsx")); //Load excel file
        XSSFWorkbook workbook = new XSSFWorkbook(fis);

        XSSFSheet sheet = workbook.getSheetAt(0);

        int lastRowNum = sheet.getLastRowNum(); //get last row number

        XSSFRow newRow = sheet.createRow(lastRowNum + 1); //go to next row

        //Create values for each cell
        XSSFCell cell1 = newRow.createCell(0);
        cell1.setCellValue(i);
        XSSFCell cell2 = newRow.createCell(1);
        cell2.setCellValue(trainingMSE);
        XSSFCell cell3 = newRow.createCell(2);
        cell3.setCellValue(testingMSE);

        //Save changes
        FileOutputStream fos = new FileOutputStream("ErrorPlot.xlsx");
        workbook.write(fos);
        fos.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

	A	B	C
1	Epochs	Training Error	Validation Error
2			
3			
4			
5			
6			

It will append training error rate and validation error once every 500 epochs

I can then plot this data and investigate when overtraining starts to occur



e.g. E_1 is the no of epochs where overtraining starts to occur with any more epochs after that

It is also the minima MSE for the validation data set, so E_1 epochs will produce the most accurate values for unseen data.

```
Basic b = new Basic(5,3,100000,0.1);
b.run();
```

I will run the algorithm with 3 hidden nodes for 100000 epochs as I expect overtraining to occur before that

```
public void createNetwork(){
    //create the weights from the input to hidden nodes
    Random rnd = new Random(1);
    for (int i = 0; i < noInput; i++) {
        for (int j = 0; j < noHidden; j++) {
            this.weights[i][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
        }
    }

    //create the weights from hidden to output node
    for (int j = 0; j < noHidden; j++) {
        this.weights[noInput][j] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput);
    }

    //create the biases for the hiddens and output node
    for (int i = 0; i < noHidden + 1; i++) {
        this.biases[i] = rnd.nextDouble() * (4.0 / noInput) - (2.0 / noInput); //random value between [-2/n,2/n]
    }
}
```

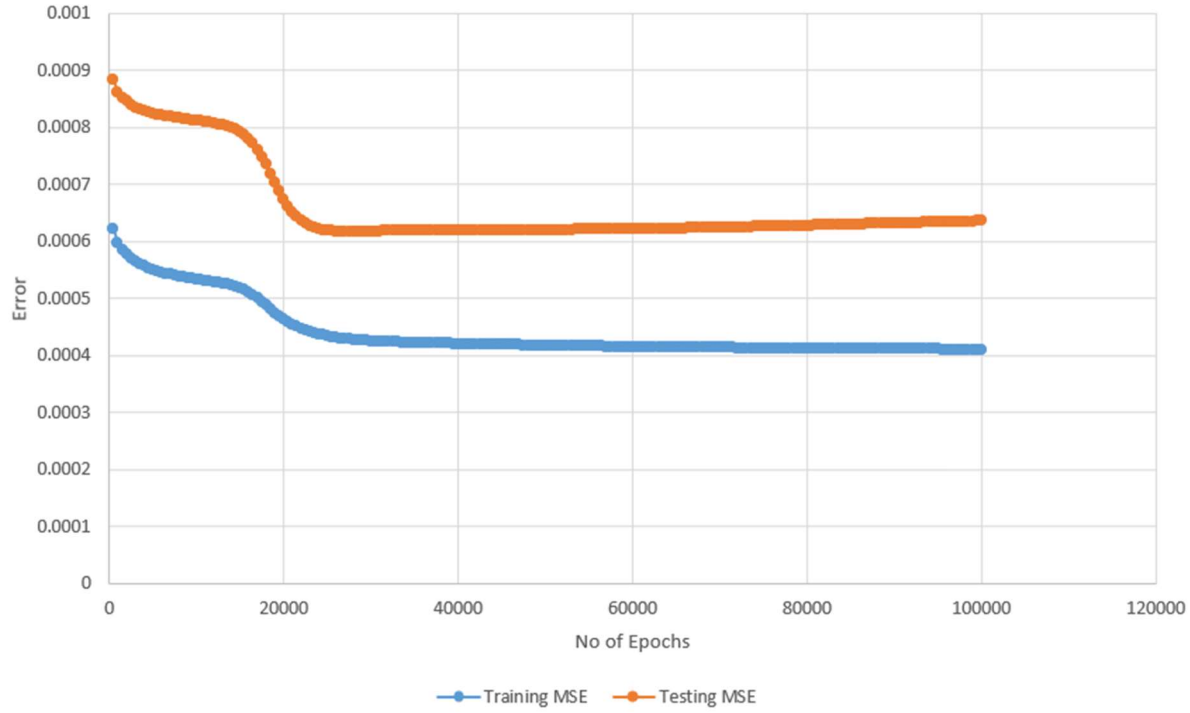
No of Epochs	Training MSE	Testing MSE
500	0.000621035	0.000883004
1000	0.000597689	0.000862135
1500	0.000585521	0.000852808
2000	0.000577109	0.000845844
2500	0.000570559	0.000840101
3000	0.00056518	0.000835378
3500	0.000560675	0.000831579
4000	0.000556861	0.000828556
4500	0.000553604	0.000826139
5000	0.000550795	0.000824172
5500	0.000548345	0.000822523
6000	0.000546182	0.000821094
6500	0.000544248	0.000819812
7000	0.000542495	0.000818626
7500	0.000540886	0.0008175
8000	0.00053939	0.000816411
8500	0.000537983	0.000815343
9000	0.000536646	0.000814286
9500	0.000535361	0.000813229
10000	0.000534114	0.000812162
10500	0.00053289	0.000811071
11000	0.000531672	0.000809937
11500	0.000530439	0.000808731
12000	0.000529166	0.000807415
12500	0.000527819	0.000805932
13000	0.000526354	0.000804208
13500	0.000524714	0.000802138
14000	0.000522829	0.000799587
14500	0.000520609	0.000796378
15000	0.000517952	0.000792289
15500	0.000514742	0.000787054
16000	0.00051087	0.000780371
16500	0.000506248	0.000771942
17000	0.000500852	0.000761536
17500	0.000494761	0.000749092
18000	0.000488179	0.000734834
18500	0.000481427	0.000719337
19000	0.000474866	0.00070347
19500	0.000468788	0.000688187
20000	0.000463347	0.00067428

20500	0.000458561	0.000662217
21000	0.000454364	0.000652135
21500	0.00045067	0.000643928
22000	0.0004474	0.000637367
22500	0.000444494	0.000632187
23000	0.000441908	0.000628137
23500	0.000439608	0.000624998
24000	0.000437566	0.000622594
24500	0.000435758	0.000620779
25000	0.000434159	0.000619437
25500	0.000432748	0.000618476
26000	0.000431504	0.000617819
26500	0.000430408	0.000617403
27000	0.000429441	0.000617177
27500	0.000428588	0.000617098
28000	0.000427833	0.000617131
28500	0.000427164	0.000617247
29000	0.000426568	0.000617423
29500	0.000426036	0.00061764
30000	0.000425558	0.000617882
30500	0.000425126	0.000618138
31000	0.000424735	0.000618398
31500	0.000424378	0.000618656
32000	0.00042405	0.000618906
32500	0.000423746	0.000619143
33000	0.000423465	0.000619366
33500	0.000423201	0.000619573
34000	0.000422952	0.000619763
34500	0.000422718	0.000619935
35000	0.000422494	0.000620089
35500	0.00042228	0.000620227
36000	0.000422074	0.000620348
36500	0.000421876	0.000620453
37000	0.000421684	0.000620544
37500	0.000421497	0.000620621
38000	0.000421315	0.000620686
38500	0.000421136	0.000620738
39000	0.000420962	0.000620781
39500	0.00042079	0.000620814
40000	0.000420621	0.000620839
40500	0.000420454	0.000620856
41000	0.00042029	0.000620866
41500	0.000420127	0.000620871
42000	0.000419967	0.000620872

42500	0.000419808	0.000620868
43000	0.00041965	0.000620861
43500	0.000419494	0.000620851
44000	0.00041934	0.00062084
44500	0.000419187	0.000620828
45000	0.000419035	0.000620814
45500	0.000418885	0.000620801
46000	0.000418736	0.000620789
46500	0.000418588	0.000620777
47000	0.000418442	0.000620766
47500	0.000418298	0.000620757
48000	0.000418155	0.000620751
48500	0.000418013	0.000620747
49000	0.000417874	0.000620746
49500	0.000417735	0.000620748
50000	0.000417599	0.000620753
50500	0.000417464	0.000620762
51000	0.000417331	0.000620776
51500	0.0004172	0.000620793
52000	0.000417071	0.000620814
52500	0.000416944	0.00062084
53000	0.000416819	0.00062087
53500	0.000416696	0.000620905
54000	0.000416574	0.000620945
54500	0.000416455	0.00062099
55000	0.000416338	0.000621039
55500	0.000416223	0.000621094
56000	0.00041611	0.000621153
56500	0.000415999	0.000621217
57000	0.00041589	0.000621286
57500	0.000415783	0.00062136
58000	0.000415678	0.000621439
58500	0.000415575	0.000621523
59000	0.000415474	0.000621611
59500	0.000415375	0.000621704
60000	0.000415278	0.000621801
60500	0.000415183	0.000621903
61000	0.00041509	0.000622009
61500	0.000414999	0.000622119
62000	0.000414909	0.000622234
62500	0.000414822	0.000622352
63000	0.000414735	0.000622474
63500	0.000414651	0.0006226
64000	0.000414568	0.00062273

64500	0.000414487	0.000622863
65000	0.000414407	0.000623
65500	0.000414329	0.000623139
66000	0.000414252	0.000623282
66500	0.000414177	0.000623428
67000	0.000414103	0.000623576
67500	0.00041403	0.000623728
68000	0.000413958	0.000623881
68500	0.000413888	0.000624038
69000	0.000413819	0.000624196
69500	0.000413751	0.000624357
70000	0.000413684	0.00062452
70500	0.000413618	0.000624685
71000	0.000413552	0.000624852
71500	0.000413488	0.000625021
72000	0.000413425	0.000625192
72500	0.000413362	0.000625364
73000	0.000413301	0.000625537
73500	0.00041324	0.000625713
74000	0.00041318	0.000625889
74500	0.00041312	0.000626067
75000	0.000413062	0.000626246
75500	0.000413003	0.000626426
76000	0.000412946	0.000626608
76500	0.000412889	0.00062679
77000	0.000412833	0.000626973
77500	0.000412777	0.000627157
78000	0.000412721	0.000627342
78500	0.000412666	0.000627528
79000	0.000412612	0.000627715
79500	0.000412558	0.000627902
80000	0.000412504	0.00062809
80500	0.000412451	0.000628279
81000	0.000412398	0.000628468
81500	0.000412346	0.000628658
82000	0.000412294	0.000628849
82500	0.000412242	0.000629039
83000	0.00041219	0.000629231
83500	0.000412139	0.000629423
84000	0.000412088	0.000629615
84500	0.000412037	0.000629807
85000	0.000411987	0.00063
85500	0.000411937	0.000630194
86000	0.000411887	0.000630387

86500	0.000411837	0.000630581
87000	0.000411787	0.000630776
87500	0.000411738	0.00063097
88000	0.000411689	0.000631165
88500	0.00041164	0.00063136
89000	0.000411591	0.000631555
89500	0.000411542	0.000631751
90000	0.000411493	0.000631947
90500	0.000411445	0.000632143
91000	0.000411396	0.000632339
91500	0.000411348	0.000632535
92000	0.0004113	0.000632732
92500	0.000411252	0.000632929
93000	0.000411204	0.000633126
93500	0.000411156	0.000633323
94000	0.000411108	0.00063352
94500	0.00041106	0.000633718
95000	0.000411013	0.000633915
95500	0.000410965	0.000634113
96000	0.000410918	0.000634311
96500	0.00041087	0.000634509
97000	0.000410823	0.000634707
97500	0.000410775	0.000634906
98000	0.000410728	0.000635104
98500	0.000410681	0.000635303
99000	0.000410634	0.000635501
99500	0.000410586	0.0006357
100000	0.000410539	0.000635899



I observed that around 28,000 epochs is when this model started overtraining. So 27,500 epochs is the turning point for the Testing MSE, which is the point just before in which overtraining starts to occur, therefore the minimum MSE this model can reach

So I will run 3 hidden nodes for 27,500 as that is the point just before overtraining occurs, therefore giving us the lowest possible MSE for Testing Data, providing the most accurate possible data for this modification

And since I've kept the same seed number as from when I ran the algorithm above, it will have the exact same weight values and MSE from when that graph was at 27,500 epochs.

```
Basic b = new Basic(5,3,27500,0.1);
b.run();
```

```
public void run(){
    for (int i = 0; i < noEpochs+1; i++) {
        if(i % 500 != 0){ //Training cycle
            training();
            //printWeights(weights,biases);
        }else{ //Validation cycle
            //validation();
            plotTrainingVsValidation(i);
        }
    }
    testing();
}
```

```
public void testing(){
    double[] predictedOutputs = new double[testingData.size()];
    double[] actualOutputs = new double[testingData.size()];
    for (int j = 0; j < testingData.size(); j++) {
        double[] u = new double[noInput+noHidden+1]; //creates new u object
        for (int k = 0; k < noInput; k++) {
            u[k] = testingData.get(j).get(k); //initializes all of the input u values
        }
        predictedOutputs[j] = deStandardiseOutput(forwardPass(u)); //set the predicted output to whatever final u value is returned on the forward pass
        actualOutputs[j]=deStandardiseOutput(testingData.get(j).get(testingData.get(j).size()-1));
    }
    double total = 0;
    //Mean Squared Error calculation
    for (int j = 0; j < testingData.size(); j++) {
        total += Math.pow((predictedOutputs[j]-actualOutputs[j]),2); //(predicted value - actual value)^2
    }
    mse = total/testingData.size(); //update MSE
    //System.out.println("MSE: " + mse);
    System.out.println("Testing MSE: " +(total/testingData.size()));
    plotPredVsActual(testingData,predictedOutputs,actualOutputs);
}
```

```
public void plotPredVsActual(ArrayList<ArrayList<Double>> data, double[] predictedOutputs, double[] actualOutputs){
    //plot data
    double[][] predVsActual = new double[data.size()][2];
    for (int i = 0; i < data.size(); i++) {
        predVsActual[i][0] = predictedOutputs[i];
        predVsActual[i][1] = actualOutputs[i];
    }
    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet sheet = workbook.createSheet("Data");

    int rowNum = 0;
    for (double[] rowData : predVsActual) {
      Row row = sheet.createRow(rowNum++);

      int colNum = 0;
      for (double field : rowData) {
        Cell cell = row.createCell(colNum++);
        cell.setCellValue(field);
      }
    }
    try (FileOutputStream outputStream = new FileOutputStream("PredVsActual.xlsx")) {
        workbook.write(outputStream);
    }catch (IOException e) {
        e.printStackTrace();
      }
}
```

Testing MSE: 6.170978211398659E-4

```
Weights:
[[-0.3422541036876316, -2.297058188979759, 0.7252570485038621]
[0.9934466495000334, -1.0361580763562281, 2.5728053359062493]
[-1.608288671200732, -2.926803096636366, 0.4444223221718855]
[0.3331896744590134, -0.6119788232671652, -0.10035190918557782]
[3.2393368215234544, 2.519631329181847, 0.06225855650469727]
[-2.9308252646886706, -4.830314042912574, 3.9185518411660607]]

Biases:
[-2.685169022748303, 4.990495238741438, -0.533869809566379, 1.934476712023877]
```

Predicted Output	Actual Output
0.278434	0.27
0.627348	0.63
0.301014	0.3
0.675616	0.65
0.551062	0.55
0.558822	0.52
0.305155	0.3

0.582843	0.59
0.246535	0.29
0.663498	0.65
0.217239	0.2
0.798333	0.8
0.653157	0.66
0.743754	0.75
0.646886	0.65
0.527683	0.55
0.7165	0.73
0.623793	0.64
0.3431	0.33
0.487185	0.51
0.282717	0.31
0.393297	0.41
0.676508	0.69
0.630225	0.61
0.429894	0.45
0.176268	0.15
0.412014	0.43
0.557116	0.53
0.342089	0.32
0.592004	0.61
0.344549	0.36
0.276278	0.28
0.640641	0.64
0.498766	0.48
0.258167	0.26
0.543098	0.54
0.438269	0.42
0.502716	0.52
0.578304	0.59
0.53332	0.53
0.221184	0.23
0.276563	0.27
0.556297	0.55
0.710841	0.72
0.346484	0.35
0.556697	0.56
0.847828	0.82
0.20638	0.22
0.246009	0.24
0.767736	0.74
0.315536	0.29

0.498926	0.52
0.618744	0.62
0.672775	0.68
0.321206	0.36
0.561598	0.58
0.465063	0.46
0.347132	0.33
0.265174	0.26
0.458003	0.44
0.550818	0.56
0.305887	0.27
0.782334	0.77
0.442103	0.43
0.518356	0.52
0.447517	0.51
0.600423	0.62
0.337906	0.36
0.304857	0.29
0.761144	0.75
0.673849	0.7
0.576878	0.58
0.489544	0.5
0.694291	0.68
0.31776	0.35
0.351992	0.33
0.512466	0.5
0.318635	0.32
0.341751	0.34
0.327319	0.32
0.398956	0.37
0.445984	0.48
0.226837	0.21
0.384962	0.36
0.553817	0.56
0.247818	0.24
0.730248	0.71
0.229681	0.23
0.614006	0.61
0.985437	0.97
0.322547	0.3
0.704617	0.68
0.230448	0.22
0.442935	0.44
0.991763	0.97

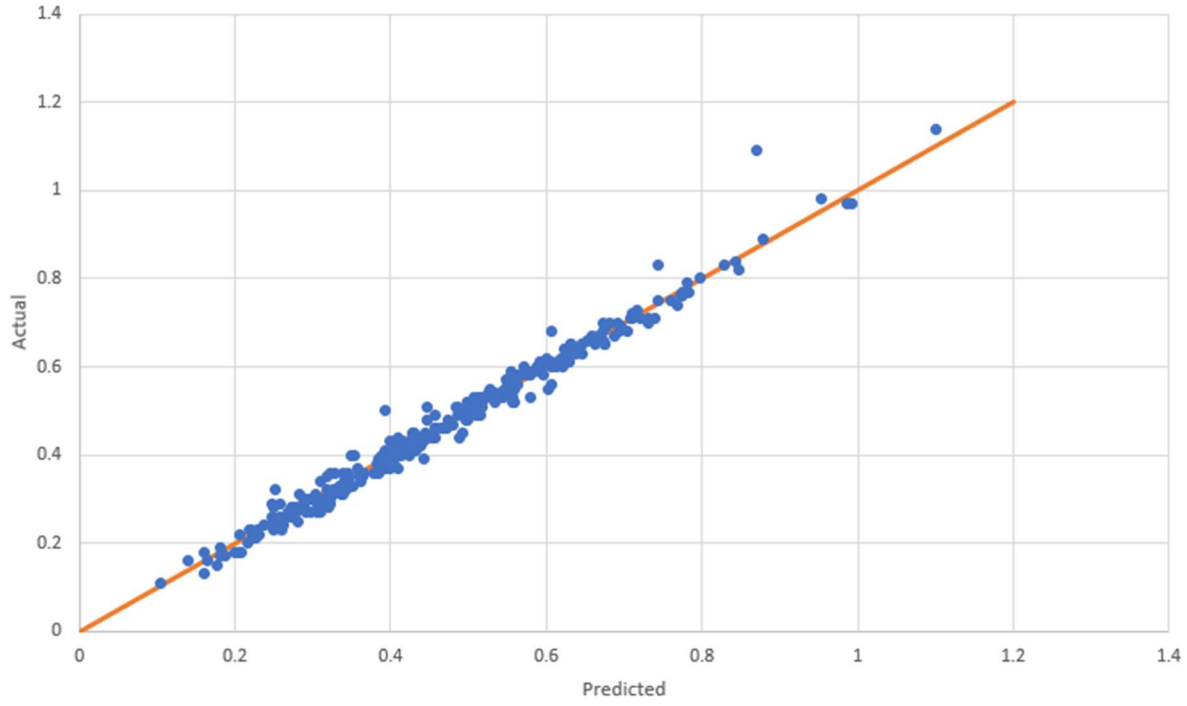
0.590965	0.6
0.359739	0.35
0.771714	0.76
0.444409	0.45
0.164745	0.16
0.505417	0.49
0.416016	0.41
0.507985	0.53
0.843331	0.84
0.103782	0.11
0.493002	0.45
0.780225	0.79
0.297879	0.3
0.549787	0.56
0.247564	0.26
0.423876	0.4
0.487191	0.49
0.260123	0.23
0.673693	0.66
0.605723	0.61
0.392548	0.37
0.470943	0.46
0.513358	0.53
0.640447	0.64
0.392926	0.5
0.56648	0.58
0.267467	0.27
0.510084	0.49
0.62157	0.62
0.378551	0.36
0.731319	0.7
0.322241	0.29
0.72162	0.71
0.870549	1.09
0.607335	0.68
0.435466	0.44
0.356839	0.37
0.587652	0.6
0.50617	0.5
0.223365	0.22
0.399648	0.43
0.9525	0.98
0.309371	0.27
0.273162	0.28

0.542876	0.54
0.543636	0.53
0.740181	0.71
0.19939	0.18
0.392912	0.4
0.207512	0.18
0.187973	0.17
0.499184	0.5
0.775307	0.77
0.309747	0.34
1.101243	1.14
0.505205	0.51
0.554756	0.59
0.632338	0.65
0.341104	0.34
0.591049	0.61
0.532651	0.54
0.412968	0.4
0.257049	0.29
0.36443	0.35
0.441698	0.39
0.428715	0.44
0.68732	0.67
0.501516	0.5
0.313572	0.3
0.390593	0.39
0.349569	0.4
0.393681	0.39
0.249802	0.28
0.775552	0.76
0.221805	0.21
0.464504	0.46
0.287729	0.29
0.579349	0.58
0.278236	0.28
0.591187	0.59
0.596056	0.58
0.467369	0.46
0.457072	0.49
0.404725	0.39
0.45301	0.44
0.416309	0.43
0.237706	0.24
0.562395	0.56

0.516175	0.53
0.33784	0.31
0.334297	0.33
0.408512	0.43
0.441171	0.43
0.182034	0.19
0.472981	0.48
0.489171	0.44
0.161089	0.13
0.622009	0.6
0.680952	0.7
0.336675	0.32
0.181681	0.17
0.549024	0.57
0.59248	0.6
0.159717	0.18
0.674949	0.65
0.480574	0.47
0.67258	0.66
0.525826	0.54
0.658587	0.67
0.388421	0.4
0.409616	0.37
0.646576	0.63
0.555978	0.52
0.409842	0.44
0.42479	0.42
0.250473	0.25
0.252358	0.32
0.559158	0.55
0.43284	0.41
0.290233	0.27
0.622645	0.62
0.304214	0.31
0.69502	0.69
0.390767	0.37
0.472124	0.46
0.645623	0.65
0.638534	0.63
0.273194	0.26
0.20437	0.18
0.327718	0.36
0.298239	0.27
0.280548	0.28

0.484274	0.51
0.497242	0.48
0.828632	0.83
0.512788	0.51
0.560782	0.57
0.710284	0.71
0.612405	0.6
0.319478	0.28
0.428227	0.43
0.402688	0.42
0.498622	0.52
0.274037	0.27
0.361952	0.34
0.223348	0.21
0.474097	0.47
0.607266	0.56
0.365954	0.36
0.457	0.46
0.625043	0.64
0.384954	0.39
0.607338	0.6
0.401291	0.4
0.62726	0.64
0.625768	0.63
0.257238	0.24
0.289489	0.28
0.547222	0.55
0.534097	0.52
0.514844	0.49
0.408907	0.41
0.281362	0.25
0.879298	0.89
0.250192	0.23
0.261396	0.24
0.290109	0.3
0.578808	0.53
0.217728	0.23
0.139322	0.16
0.601886	0.55
0.570436	0.6
0.478633	0.47
0.428365	0.45
0.521209	0.53
0.382769	0.38

0.353728	0.4
0.426474	0.41
0.521871	0.53
0.575894	0.58
0.393964	0.39
0.337471	0.31
0.691378	0.7
0.708313	0.71
0.456112	0.46
0.300379	0.3
0.532697	0.54
0.594018	0.59
0.743929	0.83
0.259961	0.25
0.326775	0.31
0.631705	0.65
0.666592	0.67
0.334465	0.33
0.516994	0.51



4 Hidden Nodes

```
Basic b = new Basic(5,4,100000,0.1);
b.run();
```

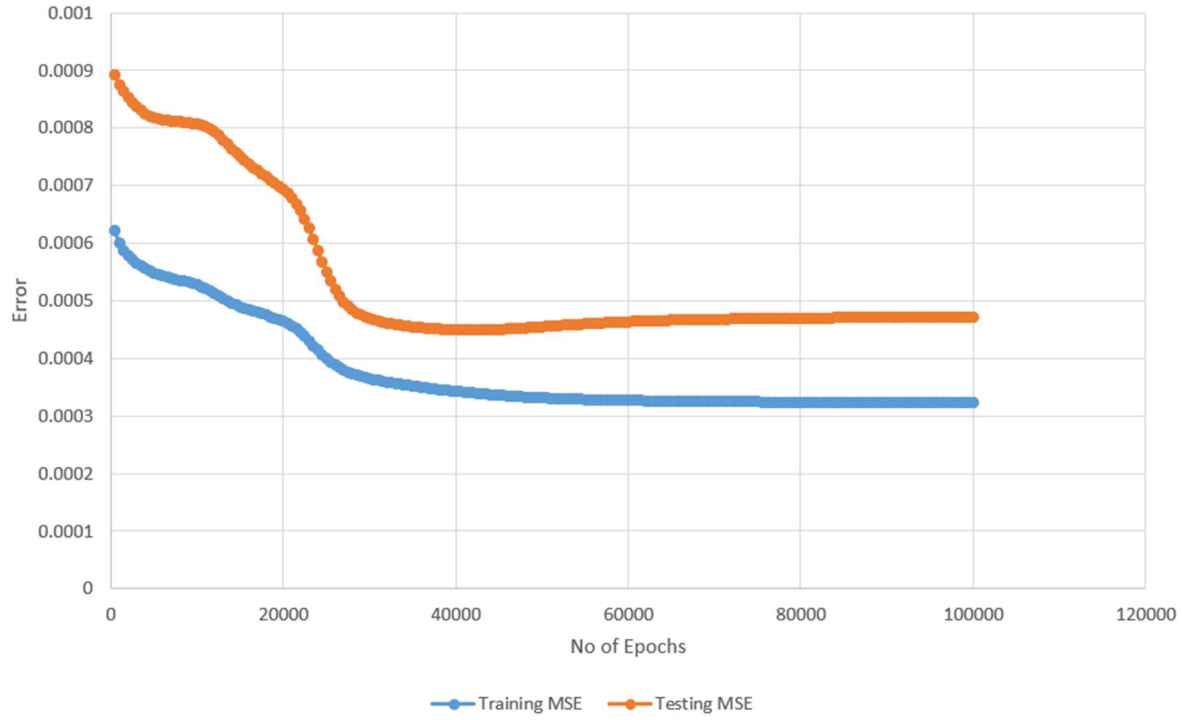
No of Epochs	Training MSE	Testing MSE
500	0.000622114	0.000891999
1000	0.000601015	0.000874213
1500	0.000588174	0.000863214
2000	0.000578866	0.000853743
2500	0.000571398	0.000844917
3000	0.000565136	0.00083696
3500	0.000559877	0.000830279
4000	0.000555487	0.000825018
4500	0.000551816	0.000821048
5000	0.000548711	0.000818117
5500	0.000546034	0.000815959
6000	0.000543669	0.00081435
6500	0.000541525	0.000813118
7000	0.000539525	0.000812135
7500	0.000537606	0.000811308
8000	0.000535714	0.000810563
8500	0.000533799	0.000809834
9000	0.000531805	0.000809051
9500	0.00052967	0.000808113
10000	0.000527306	0.000806874
10500	0.000524603	0.000805105
11000	0.000521437	0.0008025
11500	0.000517725	0.000798734
12000	0.000513508	0.000793612
12500	0.000508985	0.000787231
13000	0.000504441	0.000779962
13500	0.000500123	0.000772277
14000	0.000496172	0.000764577
14500	0.000492627	0.000757126
15000	0.000489458	0.00075006
15500	0.000486606	0.000743423
16000	0.000484002	0.000737202
16500	0.000481583	0.000731349
17000	0.000479289	0.000725799
17500	0.000477065	0.000720475
18000	0.000474853	0.000715285
18500	0.00047259	0.00071012
19000	0.000470196	0.000704843

19500	0.000467573	0.000699278
20000	0.000464598	0.000693195
20500	0.000461115	0.000686291
21000	0.000456945	0.000678188
21500	0.000451901	0.000668447
22000	0.000445843	0.000656628
22500	0.000438748	0.000642421
23000	0.000430803	0.000625832
23500	0.000422416	0.000607326
24000	0.000414118	0.000587815
24500	0.000406368	0.000568426
25000	0.000399429	0.000550195
25500	0.000393369	0.000533845
26000	0.00038814	0.000519734
26500	0.000383651	0.000507907
27000	0.000379799	0.000498201
27500	0.000376482	0.000490343
28000	0.000373606	0.000484026
28500	0.000371089	0.000478953
29000	0.000368859	0.000474862
29500	0.000366859	0.000471531
30000	0.00036504	0.000468785
30500	0.000363367	0.000466484
31000	0.00036181	0.000464521
31500	0.000360349	0.000462817
32000	0.000358968	0.000461312
32500	0.000357655	0.000459964
33000	0.0003564	0.000458742
33500	0.000355198	0.000457626
34000	0.000354043	0.0004566
34500	0.000352931	0.000455656
35000	0.000351858	0.000454785
35500	0.000350821	0.000453984
36000	0.000349818	0.00045325
36500	0.000348846	0.000452582
37000	0.000347903	0.000451977
37500	0.000346987	0.000451435
38000	0.000346097	0.000450956
38500	0.00034523	0.000450538
39000	0.000344385	0.000450182
39500	0.000343563	0.000449887
40000	0.000342761	0.000449651
40500	0.000341979	0.000449476
41000	0.000341218	0.000449359

41500	0.000340477	0.000449299
42000	0.000339757	0.000449296
42500	0.000339058	0.000449348
43000	0.000338381	0.000449453
43500	0.000337727	0.000449609
44000	0.000337095	0.000449815
44500	0.000336486	0.000450066
45000	0.000335902	0.000450361
45500	0.000335341	0.000450695
46000	0.000334806	0.000451067
46500	0.000334294	0.000451471
47000	0.000333807	0.000451905
47500	0.000333344	0.000452363
48000	0.000332904	0.000452844
48500	0.000332488	0.000453341
49000	0.000332093	0.000453852
49500	0.000331719	0.000454373
50000	0.000331366	0.0004549
50500	0.000331032	0.000455431
51000	0.000330716	0.000455962
51500	0.000330417	0.00045649
52000	0.000330134	0.000457013
52500	0.000329866	0.000457529
53000	0.000329612	0.000458035
53500	0.000329371	0.000458531
54000	0.000329142	0.000459016
54500	0.000328924	0.000459487
55000	0.000328717	0.000459944
55500	0.00032852	0.000460388
56000	0.000328331	0.000460816
56500	0.000328151	0.00046123
57000	0.000327978	0.000461628
57500	0.000327813	0.000462012
58000	0.000327655	0.000462381
58500	0.000327502	0.000462735
59000	0.000327356	0.000463075
59500	0.000327215	0.000463401
60000	0.00032708	0.000463714
60500	0.000326949	0.000464014
61000	0.000326823	0.000464301
61500	0.000326701	0.000464577
62000	0.000326584	0.00046484
62500	0.00032647	0.000465093
63000	0.00032636	0.000465336

63500	0.000326253	0.000465568
64000	0.00032615	0.000465791
64500	0.000326049	0.000466005
65000	0.000325952	0.000466211
65500	0.000325858	0.000466408
66000	0.000325766	0.000466598
66500	0.000325677	0.00046678
67000	0.00032559	0.000466955
67500	0.000325506	0.000467124
68000	0.000325424	0.000467286
68500	0.000325344	0.000467443
69000	0.000325266	0.000467593
69500	0.00032519	0.000467739
70000	0.000325116	0.000467879
70500	0.000325044	0.000468015
71000	0.000324973	0.000468146
71500	0.000324904	0.000468272
72000	0.000324837	0.000468395
72500	0.000324771	0.000468514
73000	0.000324707	0.000468628
73500	0.000324645	0.00046874
74000	0.000324583	0.000468848
74500	0.000324523	0.000468952
75000	0.000324464	0.000469054
75500	0.000324407	0.000469152
76000	0.000324351	0.000469248
76500	0.000324295	0.000469341
77000	0.000324241	0.000469432
77500	0.000324188	0.00046952
78000	0.000324136	0.000469606
78500	0.000324085	0.000469689
79000	0.000324035	0.000469771
79500	0.000323986	0.00046985
80000	0.000323938	0.000469927
80500	0.00032389	0.000470002
81000	0.000323844	0.000470076
81500	0.000323798	0.000470148
82000	0.000323753	0.000470218
82500	0.000323709	0.000470286
83000	0.000323666	0.000470353
83500	0.000323623	0.000470419
84000	0.000323581	0.000470483
84500	0.00032354	0.000470545
85000	0.000323499	0.000470607

85500	0.000323459	0.000470667
86000	0.00032342	0.000470725
86500	0.000323381	0.000470783
87000	0.000323343	0.000470839
87500	0.000323306	0.000470895
88000	0.000323269	0.000470949
88500	0.000323232	0.000471002
89000	0.000323197	0.000471054
89500	0.000323161	0.000471106
90000	0.000323127	0.000471156
90500	0.000323092	0.000471205
91000	0.000323058	0.000471254
91500	0.000323025	0.000471301
92000	0.000322992	0.000471348
92500	0.00032296	0.000471394
93000	0.000322928	0.00047144
93500	0.000322896	0.000471484
94000	0.000322865	0.000471528
94500	0.000322835	0.000471571
95000	0.000322804	0.000471614
95500	0.000322775	0.000471656
96000	0.000322745	0.000471697
96500	0.000322716	0.000471738
97000	0.000322687	0.000471778
97500	0.000322659	0.000471817
98000	0.000322631	0.000471856
98500	0.000322604	0.000471895
99000	0.000322576	0.000471933
99500	0.000322549	0.00047197
100000	0.000322523	0.000472007



So for 4 hidden nodes, overtraining occurs at 42,500

So I will run the algorithm at 42,000 epochs

```
Basic b = new Basic(5,4,42000,0.1);
b.run();
```

Testing MSE : 6.209021370874399E-4

```
Weights:
[[0.6560851071035585, 0.003806230121942593, 0.8266984349170771, -2.321025601391349]
[1.4249324052853, -2.625435513422697, -0.4227671120878659, -0.437743934218006]
[-3.384297670986222, -0.6036914945136428, 1.6324095968422128, -2.7310531986146724]
[1.1252966678074905, 0.5361400707112156, -0.13983966421759883, -0.7946209511657653]
[-2.1372653968679796, -2.3034030990201093, -3.236252603044671, 3.412211060789947]
[3.024018637606984, -2.968882047698899, 4.096349800481396, -5.3061325429971321]]

Biases:
[-1.3528524532712671, 1.6052408196077557, 2.03131299701744, 4.2480263955796715, 2.2548700820852443]
```

Predicted Output	Actual Output
0.2773	0.27
0.629431	0.63
0.294353	0.3
0.67581	0.65
0.554088	0.55
0.540928	0.52
0.308292	0.3
0.577641	0.59
0.250651	0.29

0.648189	0.65
0.216679	0.2
0.787725	0.8
0.656828	0.66
0.734275	0.75
0.634156	0.65
0.520674	0.55
0.711214	0.73
0.618487	0.64
0.340142	0.33
0.503019	0.51
0.291725	0.31
0.389059	0.41
0.682078	0.69
0.619841	0.61
0.429141	0.45
0.177151	0.15
0.406846	0.43
0.553978	0.53
0.337118	0.32
0.592242	0.61
0.338133	0.36
0.273317	0.28
0.647582	0.64
0.489153	0.48
0.254592	0.26
0.536808	0.54
0.427089	0.42
0.510514	0.52
0.57093	0.59
0.540898	0.53
0.221357	0.23
0.271926	0.27
0.560393	0.55
0.712737	0.72
0.34402	0.35
0.564421	0.56
0.841835	0.82
0.205551	0.22
0.250622	0.24
0.766756	0.74
0.311745	0.29
0.507693	0.52
0.627623	0.62

0.667597	0.68
0.315509	0.36
0.555292	0.58
0.448983	0.46
0.322685	0.33
0.268289	0.26
0.481092	0.44
0.548371	0.56
0.301345	0.27
0.772879	0.77
0.430241	0.43
0.512333	0.52
0.487592	0.51
0.59174	0.62
0.357187	0.36
0.303689	0.29
0.763359	0.75
0.676608	0.7
0.58813	0.58
0.475046	0.5
0.682464	0.68
0.316841	0.35
0.351347	0.33
0.508482	0.5
0.312225	0.32
0.335897	0.34
0.32498	0.32
0.420077	0.37
0.473249	0.48
0.226562	0.21
0.375093	0.36
0.564278	0.56
0.24624	0.24
0.729432	0.71
0.229133	0.23
0.625621	0.61
0.978339	0.97
0.317752	0.3
0.710539	0.68
0.230539	0.22
0.443759	0.44
0.99416	0.97
0.596107	0.6
0.355822	0.35

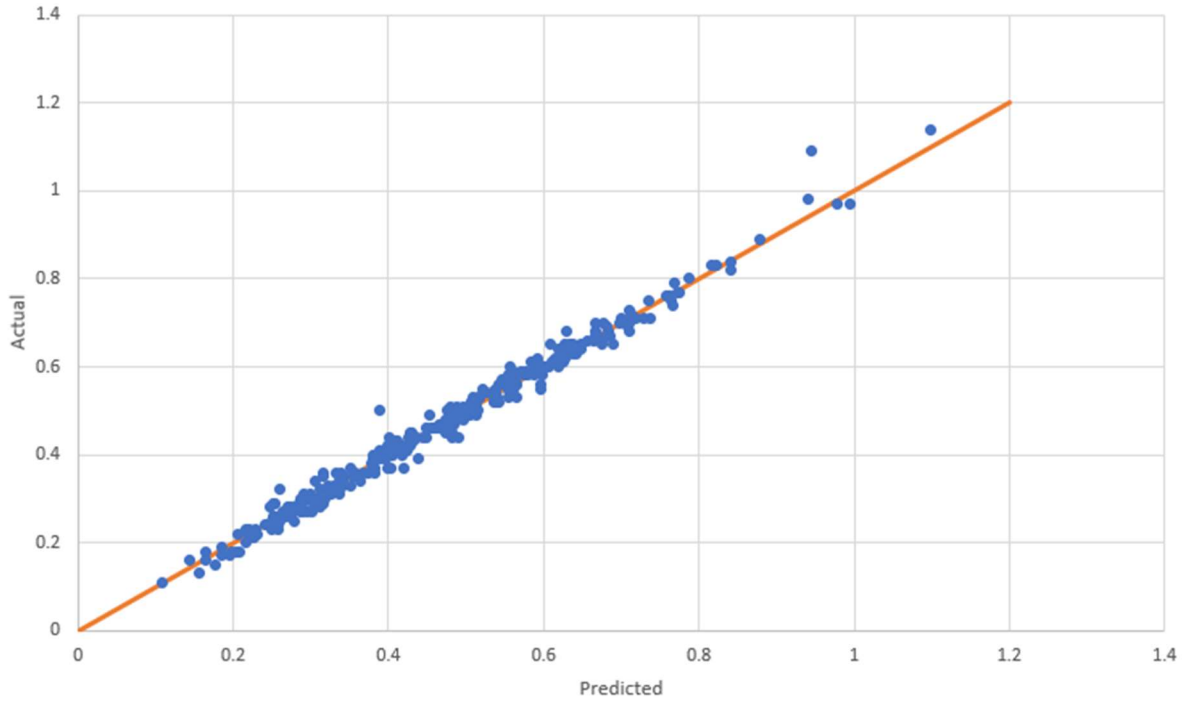
0.759008	0.76
0.473641	0.45
0.163862	0.16
0.503443	0.49
0.408569	0.41
0.509434	0.53
0.840677	0.84
0.107911	0.11
0.481672	0.45
0.769337	0.79
0.293115	0.3
0.543066	0.56
0.25144	0.26
0.417167	0.4
0.483643	0.49
0.257111	0.23
0.675183	0.66
0.611206	0.61
0.402153	0.37
0.464929	0.46
0.51749	0.53
0.640353	0.64
0.388137	0.5
0.575813	0.58
0.264252	0.27
0.505627	0.49
0.61579	0.62
0.382836	0.36
0.711173	0.7
0.316771	0.29
0.712987	0.71
0.945171	1.09
0.62884	0.68
0.436261	0.44
0.35079	0.37
0.592166	0.6
0.515943	0.5
0.220936	0.22
0.4034	0.43
0.940972	0.98
0.289202	0.27
0.270297	0.28
0.534838	0.54
0.536018	0.53

0.736684	0.71
0.197788	0.18
0.395496	0.4
0.207329	0.18
0.194815	0.17
0.48956	0.5
0.775314	0.77
0.305773	0.34
1.098058	1.14
0.496172	0.51
0.560597	0.59
0.608998	0.65
0.340488	0.34
0.584706	0.61
0.537564	0.54
0.404463	0.4
0.254365	0.29
0.356455	0.35
0.438133	0.39
0.426229	0.44
0.685862	0.67
0.50166	0.5
0.307781	0.3
0.387235	0.39
0.383105	0.4
0.389026	0.39
0.247654	0.28
0.761547	0.76
0.220533	0.21
0.470454	0.46
0.284507	0.29
0.575549	0.58
0.27847	0.28
0.591546	0.59
0.599277	0.58
0.460495	0.46
0.452685	0.49
0.397208	0.39
0.448501	0.44
0.411682	0.43
0.242291	0.24
0.555394	0.56
0.51683	0.53
0.324461	0.31

0.330249	0.33
0.406521	0.43
0.431299	0.43
0.185888	0.19
0.475599	0.48
0.490858	0.44
0.157126	0.13
0.619686	0.6
0.665797	0.7
0.330653	0.32
0.185362	0.17
0.547313	0.57
0.600112	0.6
0.165266	0.18
0.690369	0.65
0.483748	0.47
0.663854	0.66
0.52516	0.54
0.670678	0.67
0.384994	0.4
0.398164	0.37
0.638341	0.63
0.535838	0.52
0.40139	0.44
0.422692	0.42
0.248912	0.25
0.260004	0.32
0.554484	0.55
0.423115	0.41
0.287908	0.27
0.62585	0.62
0.299646	0.31
0.709042	0.69
0.382534	0.37
0.473156	0.46
0.638505	0.65
0.642352	0.63
0.274405	0.26
0.203214	0.18
0.333271	0.36
0.295153	0.27
0.280611	0.28
0.480705	0.51
0.49727	0.48

0.822181	0.83
0.51111	0.51
0.563776	0.57
0.71907	0.71
0.604512	0.6
0.312431	0.28
0.429226	0.43
0.399107	0.42
0.507006	0.52
0.272126	0.27
0.36281	0.34
0.221538	0.21
0.465226	0.47
0.596005	0.56
0.367096	0.36
0.454748	0.46
0.625117	0.64
0.381105	0.39
0.607558	0.6
0.399661	0.4
0.643174	0.64
0.621048	0.63
0.25113	0.24
0.289519	0.28
0.53885	0.55
0.542123	0.52
0.51315	0.49
0.416445	0.41
0.278927	0.25
0.878419	0.89
0.250387	0.23
0.25856	0.24
0.286394	0.3
0.564634	0.53
0.216921	0.23
0.144665	0.16
0.595351	0.55
0.556987	0.6
0.475842	0.47
0.428581	0.45
0.517449	0.53
0.377848	0.38
0.381128	0.4
0.421624	0.41

0.537277	0.53
0.580353	0.58
0.39503	0.39
0.337229	0.31
0.696898	0.7
0.699551	0.71
0.458393	0.46
0.295155	0.3
0.523953	0.54
0.58604	0.59
0.816128	0.83
0.260351	0.25
0.325544	0.31
0.627877	0.65
0.667154	0.67
0.331488	0.33
0.513021	0.51



10 Hidden Nodes

```
Basic b = new Basic(5,10,100000,0.1);
b.run();
```

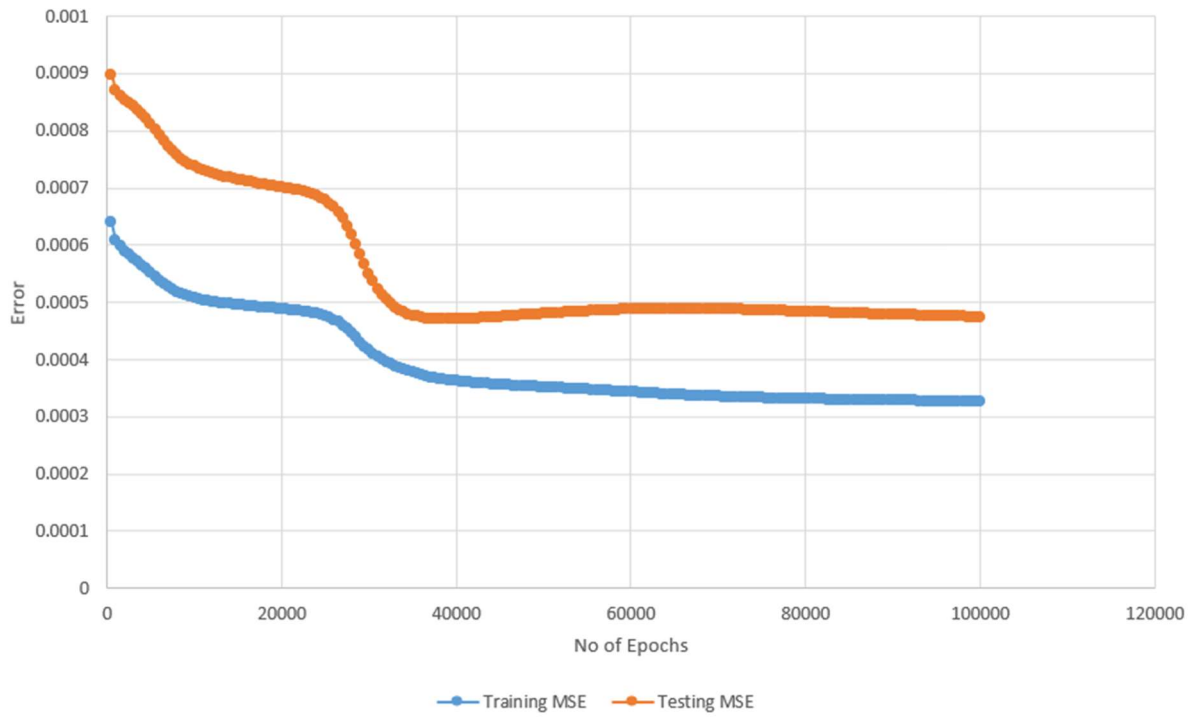
No of Epochs	Training MSE	Testing MSE
500	0.000640133	0.000898536
1000	0.000608854	0.000870721
1500	0.000597824	0.00086143
2000	0.00059043	0.000855086
2500	0.000583961	0.0008492
3000	0.000577846	0.000843218
3500	0.000571821	0.000836831
4000	0.000565654	0.000829741
4500	0.000559184	0.000821704
5000	0.00055241	0.000812664
5500	0.000545524	0.000802838
6000	0.000538852	0.000792686
6500	0.000532706	0.000782752
7000	0.000527284	0.000773488
7500	0.00052264	0.000765171
8000	0.000518717	0.000757895
8500	0.000515409	0.000751626
9000	0.000512599	0.000746255
9500	0.000510187	0.000741648
10000	0.000508091	0.000737672
10500	0.000506251	0.00073421
11000	0.000504619	0.000731161
11500	0.000503156	0.000728447
12000	0.000501835	0.000726004
12500	0.000500631	0.00072378
13000	0.000499525	0.000721737
13500	0.000498501	0.000719843
14000	0.000497545	0.000718073
14500	0.000496648	0.000716407
15000	0.0004958	0.000714831
15500	0.000494993	0.000713329
16000	0.00049422	0.000711893
16500	0.000493476	0.000710511
17000	0.000492755	0.000709174
17500	0.000492051	0.000707874
18000	0.000491359	0.000706602
18500	0.000490672	0.000705348
19000	0.000489985	0.000704102
19500	0.00048929	0.000702851
20000	0.000488578	0.000701581
20500	0.00048784	0.000700272
21000	0.000487064	0.000698903

21500	0.000486233	0.000697443
22000	0.000485329	0.000695854
22500	0.000484327	0.000694083
23000	0.000483192	0.000692061
23500	0.000481882	0.000689696
24000	0.000480337	0.000686862
24500	0.000478481	0.00068339
25000	0.00047621	0.000679057
25500	0.000473394	0.00067358
26000	0.000469876	0.000666616
26500	0.000465492	0.000657802
27000	0.000460108	0.000646829
27500	0.000453696	0.000633571
28000	0.000446411	0.000618224
28500	0.000438618	0.000601367
29000	0.00043081	0.000583884
29500	0.000423443	0.000566749
30000	0.000416786	0.000550792
30500	0.000410894	0.000536544
31000	0.000405682	0.000524228
31500	0.000401017	0.00051382
32000	0.000396781	0.000505151
32500	0.000392893	0.000497991
33000	0.0003893	0.000492105
33500	0.000385977	0.000487281
34000	0.000382909	0.00048334
34500	0.00038009	0.000480139
35000	0.000377516	0.00047756
35500	0.000375181	0.00047551
36000	0.000373076	0.000473909
36500	0.000371189	0.00047269
37000	0.000369504	0.000471797
37500	0.000368004	0.000471177
38000	0.000366669	0.000470787
38500	0.000365479	0.000470588
39000	0.000364415	0.000470546
39500	0.000363458	0.000470632
40000	0.000362592	0.000470822
40500	0.000361803	0.000471094
41000	0.000361077	0.000471431
41500	0.000360404	0.000471819
42000	0.000359774	0.000472246
42500	0.00035918	0.000472704
43000	0.000358616	0.000473183

43500	0.000358075	0.000473679
44000	0.000357555	0.000474186
44500	0.000357051	0.000474701
45000	0.000356562	0.000475221
45500	0.000356084	0.000475742
46000	0.000355616	0.000476264
46500	0.000355156	0.000476785
47000	0.000354703	0.000477303
47500	0.000354257	0.000477818
48000	0.000353815	0.000478329
48500	0.000353378	0.000478835
49000	0.000352945	0.000479336
49500	0.000352515	0.000479831
50000	0.000352088	0.00048032
50500	0.000351663	0.000480802
51000	0.00035124	0.000481278
51500	0.000350819	0.000481746
52000	0.000350398	0.000482206
52500	0.000349978	0.000482659
53000	0.000349558	0.000483103
53500	0.000349138	0.000483539
54000	0.000348717	0.000483965
54500	0.000348294	0.000484382
55000	0.000347871	0.000484789
55500	0.000347445	0.000485186
56000	0.000347016	0.000485572
56500	0.000346585	0.000485946
57000	0.000346151	0.000486308
57500	0.000345713	0.000486656
58000	0.000345272	0.000486992
58500	0.000344827	0.000487313
59000	0.000344378	0.000487618
59500	0.000343927	0.000487906
60000	0.000343473	0.000488178
60500	0.000343017	0.00048843
61000	0.000342561	0.000488662
61500	0.000342105	0.000488873
62000	0.000341652	0.000489061
62500	0.000341203	0.000489224
63000	0.00034076	0.000489363
63500	0.000340324	0.000489475
64000	0.000339896	0.000489561
64500	0.000339477	0.000489619
65000	0.000339069	0.000489651

65500	0.000338671	0.000489657
66000	0.000338285	0.000489638
66500	0.00033791	0.000489594
67000	0.000337547	0.000489528
67500	0.000337196	0.000489441
68000	0.000336856	0.000489335
68500	0.000336527	0.00048921
69000	0.000336209	0.000489069
69500	0.000335902	0.000488912
70000	0.000335606	0.000488742
70500	0.000335319	0.000488559
71000	0.000335042	0.000488366
71500	0.000334775	0.000488162
72000	0.000334517	0.000487949
72500	0.000334268	0.000487729
73000	0.000334027	0.000487501
73500	0.000333794	0.000487268
74000	0.000333568	0.00048703
74500	0.000333351	0.000486787
75000	0.00033314	0.00048654
75500	0.000332936	0.00048629
76000	0.000332738	0.000486038
76500	0.000332546	0.000485783
77000	0.00033236	0.000485526
77500	0.00033218	0.000485268
78000	0.000332004	0.000485009
78500	0.000331834	0.00048475
79000	0.000331668	0.00048449
79500	0.000331507	0.000484229
80000	0.00033135	0.000483969
80500	0.000331197	0.000483708
81000	0.000331047	0.000483448
81500	0.000330901	0.000483189
82000	0.000330759	0.00048293
82500	0.000330619	0.000482671
83000	0.000330482	0.000482414
83500	0.000330349	0.000482158
84000	0.000330217	0.000481902
84500	0.000330088	0.000481648
85000	0.000329962	0.000481395
85500	0.000329837	0.000481143
86000	0.000329715	0.000480892
86500	0.000329594	0.000480643
87000	0.000329476	0.000480395

87500	0.000329359	0.000480149
88000	0.000329243	0.000479904
88500	0.000329129	0.000479661
89000	0.000329017	0.00047942
89500	0.000328905	0.000479181
90000	0.000328795	0.000478943
90500	0.000328686	0.000478707
91000	0.000328578	0.000478473
91500	0.000328472	0.000478241
92000	0.000328366	0.000478011
92500	0.000328261	0.000477783
93000	0.000328157	0.000477558
93500	0.000328054	0.000477334
94000	0.000327951	0.000477112
94500	0.000327849	0.000476893
95000	0.000327748	0.000476676
95500	0.000327648	0.000476462
96000	0.000327548	0.000476249
96500	0.000327449	0.000476039
97000	0.00032735	0.000475832
97500	0.000327252	0.000475627
98000	0.000327155	0.000475424
98500	0.000327058	0.000475224
99000	0.000326961	0.000475026
99500	0.000326865	0.000474831
100000	0.000326769	0.000474639



Overtraining occurs at 39,500 epochs

I will therefore plot the data for 39,000 epochs

```
Basic b = new Basic(5,10,39000,0.1);
b.run();
```

Testing MSE: 4.705462037127505E-4

Weights:
[[-0.020802921224949707, -0.026607247819836893, -0.21517582773574484, -2.7688907068157156, -0.32774757420595213, 0.14839098086571278, 1.1853160397513307, 0.95566706285918, -0.03653063281297816, 0.10437348507665503]
[0.5140893487402828, -0.23464722092404378, 0.5881753403603937, -2.1626665188587113, -2.854340600070492, -0.3361905141866316, -0.3317115087339858, -0.3114870087444452, -0.007228349746112805, 0.10357401341460322]
[-0.12185727758030465, -0.1283945605494088, 0.05957579720318071, -0.30245206845987055, -0.3349063628769788, -0.48812974638608275, 2.708139508985637, 1.2000785976061956, 0.23991706890170494, -0.09551569179598457]
[-0.44808917671562803, -0.29324502687361864, 0.01739131005632731, -0.8681167057690583, 0.15836219103738786, 0.009664296009621917, 0.1459130067922311, -0.07099802323352658, -0.43432913706898896, -0.23222651071043782]
[-0.5994633071027148, 0.19020116577697166, -0.8846600878822752, 3.692872327074649, -0.8244437459654212, 0.5635218182573697, -0.5319416336201733, -3.612116850688903, -0.38119811959339583, -0.26700031674337155]
[0.8575064999931832, -0.06317337802639562, 0.9223825980868643, -4.066759309820879, -2.575502040753448, -0.5627762237209992, 3.573188314841941, 3.058063982305542, 0.5705375864029811, 0.3670472453142118]]

Biases:
[0.20557577336389732, 0.08773718726104941, 0.335957804863276, 3.9316246306812253, 0.583434710368595, -0.43167817756323973, -3.6355004562988187, 3.1352720216247643, 0.13804838863534363, -0.4164311279525243, -0.04789439289706865]

Predicted Outputs	Actual Outputs
0.279342	0.27
0.635259	0.63
0.295747	0.3
0.677797	0.65
0.55482	0.55
0.552237	0.52
0.305629	0.3
0.579702	0.59
0.256472	0.29
0.652734	0.65
0.217234	0.2
0.786691	0.8
0.662097	0.66
0.730807	0.75
0.634438	0.65
0.520901	0.55
0.724892	0.73
0.617176	0.64
0.34364	0.33
0.491036	0.51
0.28864	0.31
0.388235	0.41
0.684158	0.69
0.629942	0.61
0.426345	0.45
0.176613	0.15
0.405713	0.43
0.553116	0.53
0.339372	0.32
0.59048	0.61
0.340394	0.36

0.271428	0.28
0.651347	0.64
0.490295	0.48
0.260666	0.26
0.535803	0.54
0.431362	0.42
0.508248	0.52
0.58053	0.59
0.538911	0.53
0.225452	0.23
0.273424	0.27
0.559482	0.55
0.715082	0.72
0.345861	0.35
0.564072	0.56
0.846516	0.82
0.212253	0.22
0.254296	0.24
0.765659	0.74
0.312977	0.29
0.508461	0.52
0.635544	0.62
0.668595	0.68
0.318241	0.36
0.554446	0.58
0.458497	0.46
0.338815	0.33
0.266238	0.26
0.467175	0.44
0.546278	0.56
0.304894	0.27
0.769406	0.77
0.43721	0.43
0.509968	0.52
0.459708	0.51
0.591981	0.62
0.35048	0.36
0.307739	0.29
0.771259	0.75
0.680023	0.7
0.58693	0.58
0.487638	0.5
0.691054	0.68
0.320067	0.35

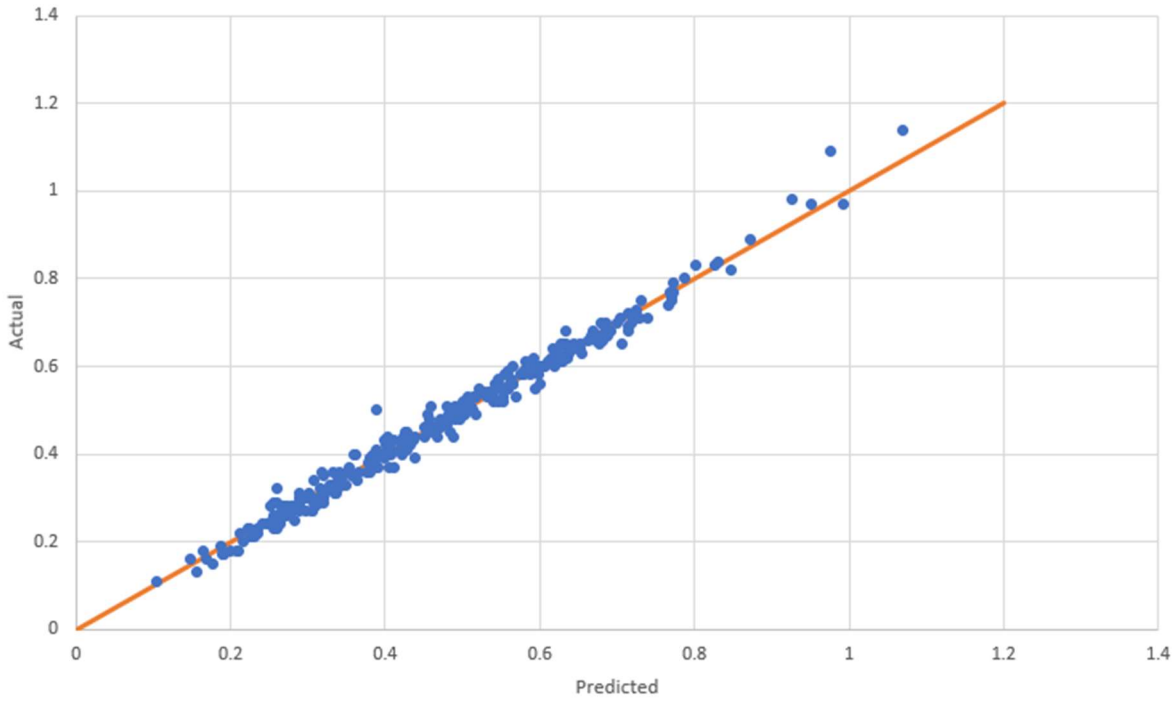
0.348955	0.33
0.507979	0.5
0.315428	0.32
0.33855	0.34
0.328357	0.32
0.411643	0.37
0.457904	0.48
0.23082	0.21
0.379918	0.36
0.564321	0.56
0.247633	0.24
0.728109	0.71
0.234494	0.23
0.628061	0.61
0.950496	0.97
0.320335	0.3
0.714138	0.68
0.2358	0.22
0.4386	0.44
0.992106	0.97
0.596843	0.6
0.357594	0.35
0.770479	0.76
0.456472	0.45
0.168738	0.16
0.502395	0.49
0.419572	0.41
0.506922	0.53
0.830066	0.84
0.105022	0.11
0.484748	0.45
0.772176	0.79
0.295951	0.3
0.541842	0.56
0.256609	0.26
0.421401	0.4
0.482031	0.49
0.260708	0.23
0.681885	0.66
0.611448	0.61
0.391717	0.37
0.465469	0.46
0.51724	0.53
0.641272	0.64

0.387857	0.5
0.576065	0.58
0.265234	0.27
0.501351	0.49
0.61389	0.62
0.376175	0.36
0.718168	0.7
0.320259	0.29
0.720222	0.71
0.976204	1.09
0.633477	0.68
0.432438	0.44
0.352579	0.37
0.591357	0.6
0.513315	0.5
0.227836	0.22
0.398949	0.43
0.92702	0.98
0.30508	0.27
0.267916	0.28
0.538083	0.54
0.537712	0.53
0.740372	0.71
0.200202	0.18
0.389192	0.4
0.210265	0.18
0.191023	0.17
0.492543	0.5
0.772492	0.77
0.307869	0.34
1.069298	1.14
0.496972	0.51
0.558158	0.59
0.632119	0.65
0.341105	0.34
0.582182	0.61
0.53474	0.54
0.408251	0.4
0.260481	0.29
0.356932	0.35
0.438123	0.39
0.424012	0.44
0.686898	0.67
0.499246	0.5

0.313203	0.3
0.385741	0.39
0.358773	0.4
0.388098	0.39
0.250865	0.28
0.771099	0.76
0.226033	0.21
0.465569	0.46
0.288913	0.29
0.575896	0.58
0.282612	0.28
0.591001	0.59
0.597812	0.58
0.46482	0.46
0.45458	0.49
0.399591	0.39
0.450626	0.44
0.410864	0.43
0.241788	0.24
0.560284	0.56
0.515451	0.53
0.337551	0.31
0.328252	0.33
0.407106	0.43
0.432744	0.43
0.186549	0.19
0.472659	0.48
0.487395	0.44
0.156749	0.13
0.618322	0.6
0.68522	0.7
0.335776	0.32
0.188705	0.17
0.545409	0.57
0.60042	0.6
0.16466	0.18
0.705516	0.65
0.47993	0.47
0.673187	0.66
0.526762	0.54
0.678475	0.67
0.384826	0.4
0.404796	0.37
0.636815	0.63

0.545661	0.52
0.402472	0.44
0.421959	0.42
0.254352	0.25
0.260428	0.32
0.558478	0.55
0.427722	0.41
0.289217	0.27
0.626579	0.62
0.302374	0.31
0.715237	0.69
0.385681	0.37
0.471402	0.46
0.644782	0.65
0.653631	0.63
0.271607	0.26
0.207446	0.18
0.332352	0.36
0.296337	0.27
0.278505	0.28
0.480564	0.51
0.495598	0.48
0.827358	0.83
0.508766	0.51
0.563409	0.57
0.723987	0.71
0.604459	0.6
0.310682	0.28
0.424872	0.43
0.40174	0.42
0.501206	0.52
0.270285	0.27
0.364045	0.34
0.226138	0.21
0.468819	0.47
0.6004	0.56
0.366411	0.36
0.450167	0.46
0.624134	0.64
0.380444	0.39
0.607062	0.6
0.396331	0.4
0.648	0.64
0.62219	0.63

0.259319	0.24
0.288638	0.28
0.544081	0.55
0.540708	0.52
0.517482	0.49
0.406794	0.41
0.282105	0.25
0.873096	0.89
0.255207	0.23
0.264675	0.24
0.28926	0.3
0.56874	0.53
0.222592	0.23
0.14756	0.16
0.594909	0.55
0.564744	0.6
0.47415	0.47
0.427378	0.45
0.515144	0.53
0.378694	0.38
0.36083	0.4
0.420066	0.41
0.532821	0.53
0.579097	0.58
0.389897	0.39
0.33391	0.31
0.699509	0.7
0.703554	0.71
0.453954	0.46
0.297446	0.3
0.523119	0.54
0.592318	0.59
0.80097	0.83
0.263086	0.25
0.322214	0.31
0.6273	0.65
0.666311	0.67
0.331753	0.33
0.510541	0.51



Decision

No of Hidden Nodes	Minima Testing MSE (The point just before overtraining starts)	No of Epochs
3	6.170978211398659E-4	27,500
4	4.493479551351621E-4	42,000
10	4.705462037127505E-4	39,000

So authough 3 hidden nodes reaches its minima much more quickly, 4 hidden nodes has a significantly more accurate prediction on unseen data after more epochs than the other 2, so 4 hidden nodes is the most optimal.

4 Inputs

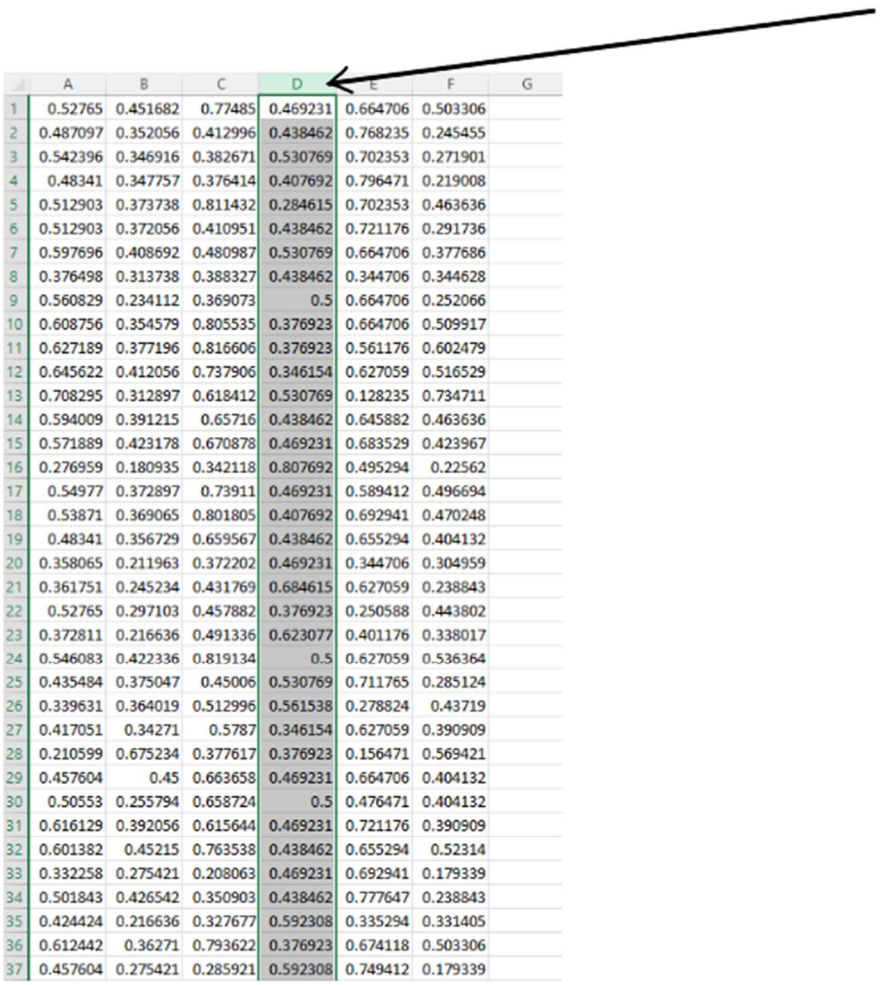
I will now test the Basic Model against the 4 Inputs, not including DSP

To achieve this using my algorithm, I needed to remove the column from the Training, Validation and Testing Data respectively and create a new Excel file for each of them

This is because my algorithm works by getting first n inputs from left to right

```
Basic b = new Basic(n,8,39000,0.1);
b.run();
```

So I will need to remove the DSP column from the dataset and create it as a new one as my algorithm cannot skip past input columns, which is a limitation of my algorithm.



	A	B	C	D	E	F	G
1	0.52765	0.451682	0.77485	0.469231	0.664706	0.503306	
2	0.487097	0.352056	0.412996	0.438462	0.768235	0.245455	
3	0.542396	0.346916	0.382671	0.530769	0.702353	0.271901	
4	0.48341	0.347757	0.376414	0.407692	0.796471	0.219008	
5	0.512903	0.373738	0.811432	0.284615	0.702353	0.463636	
6	0.512903	0.372056	0.410951	0.438462	0.721176	0.291736	
7	0.597696	0.408692	0.480987	0.530769	0.664706	0.377686	
8	0.376498	0.313738	0.388327	0.438462	0.344706	0.344628	
9	0.560829	0.234112	0.369073	0.5	0.664706	0.252066	
10	0.608756	0.354579	0.805535	0.376923	0.664706	0.509917	
11	0.627189	0.377196	0.816606	0.376923	0.561176	0.602479	
12	0.645622	0.412056	0.737906	0.346154	0.627059	0.516529	
13	0.708295	0.312897	0.618412	0.530769	0.128235	0.734711	
14	0.594009	0.391215	0.65716	0.438462	0.645882	0.463636	
15	0.571889	0.423178	0.670878	0.469231	0.683529	0.423967	
16	0.276959	0.180935	0.342118	0.807692	0.495294	0.22562	
17	0.54977	0.372897	0.73911	0.469231	0.589412	0.496694	
18	0.53871	0.369065	0.801805	0.407692	0.692941	0.470248	
19	0.48341	0.356729	0.659567	0.438462	0.655294	0.404132	
20	0.358065	0.211963	0.372202	0.469231	0.344706	0.304959	
21	0.361751	0.245234	0.431769	0.684615	0.627059	0.238843	
22	0.52765	0.297103	0.457882	0.376923	0.250588	0.443802	
23	0.372811	0.216636	0.491336	0.623077	0.401176	0.338017	
24	0.546083	0.422336	0.819134	0.5	0.627059	0.536364	
25	0.435484	0.375047	0.45006	0.530769	0.711765	0.285124	
26	0.339631	0.364019	0.512996	0.561538	0.278824	0.43719	
27	0.417051	0.34271	0.5787	0.346154	0.627059	0.390909	
28	0.210599	0.675234	0.377617	0.376923	0.156471	0.569421	
29	0.457604	0.45	0.663658	0.469231	0.664706	0.404132	
30	0.50553	0.255794	0.658724	0.5	0.476471	0.404132	
31	0.616129	0.392056	0.615644	0.469231	0.721176	0.390909	
32	0.601382	0.45215	0.763538	0.438462	0.655294	0.52314	
33	0.332258	0.275421	0.208063	0.469231	0.692941	0.179339	
34	0.501843	0.426542	0.350903	0.438462	0.777647	0.238843	
35	0.424424	0.216636	0.327677	0.592308	0.335294	0.331405	
36	0.612442	0.36271	0.793622	0.376923	0.674118	0.503306	
37	0.457604	0.275421	0.285921	0.592308	0.749412	0.179339	

```
public static void main(String[] args) {
    try {
        // read from first data set
        FileInputStream fis1 = new FileInputStream(new File("TrainingDataWoDSP.xlsx"));
        XSSFWorkbook wb1 = new XSSFWorkbook(fis1);
        XSSFSheet sheet1 = wb1.getSheetAt(0);
        for (Row row : sheet1) {
            ArrayList<Double> rowData = new ArrayList<>();
            // start from the second cell
            for (Cell cell: row) {
                if (cell.getCellType() == Cell.CELL_TYPE_NUMERIC) {
                    rowData.add(cell.getNumericCellValue());
                }
            }
            trainingData.add(rowData);
        }

        // read from second data set
        FileInputStream fis2 = new FileInputStream(new File("ValidationDataWoDSP.xlsx"));
        XSSFWorkbook wb2 = new XSSFWorkbook(fis2);
        XSSFSheet sheet2 = wb2.getSheetAt(0);
        for (Row row : sheet2) {
            ArrayList<Double> rowData = new ArrayList<>();
            // start from the second cell
            for (Cell cell: row) {
                if (cell.getCellType() == Cell.CELL_TYPE_NUMERIC) {
                    rowData.add(cell.getNumericCellValue());
                }
            }
            validationData.add(rowData);
        }

        // read from third data set
        FileInputStream fis3 = new FileInputStream(new File("TestingDataWoDSP.xlsx"));
        XSSFWorkbook wb3 = new XSSFWorkbook(fis3);
        XSSFSheet sheet3 = wb3.getSheetAt(0);
        for (Row row : sheet3) {
            ArrayList<Double> rowData = new ArrayList<>();
            // start from the second cell
            for (Cell cell: row) {
                if (cell.getCellType() == Cell.CELL_TYPE_NUMERIC) {
                    rowData.add(cell.getNumericCellValue());
                }
            }
            testingData.add(rowData);
        }

    } catch(IOException e) {
        e.printStackTrace();
    }
```

Hidden Nodes

1000 epochs

No of Epochs	Training MSE	Validation MSE
2	6.251520582288704E-4	8.717932625004327E-4
3	6.244495674004528E-4	8.648546563583967E-4
4	6.229240877511168E-4	8.744525987522818E-4
5	6.234490660491067E-4	8.700788187204682E-4
6	6.221030737388055E-4	8.589067713135355E-4
7	6.291917932046072E-4	8.740427634543981E-4
8	6.55021187101081E-4	8.854835992521751E-4
9	6.318588656144966E-4	8.750129587301779E-4
10	6.682914552024118E-4	8.951359138429954E-4

5000 Epochs

No of Epochs	Training MSE	Testing MSE
2	5.718362189445487E-4	8.270188289191708E-4
3	5.661625834643315E-4	8.293070950526097E-4
4	5.660118784248093E-4	8.187723419813083E-4
5	5.682587871913167E-4	8.288712650797146E-4
6	5.571584951240314E-4	8.208010616124493E-4
7	5.656971526498818E-4	8.217317309651557E-4
8	5.724731083214114E-4	8.175021458517798E-4
9	5.697306093905964E-4	8.205760136208847E-4
10	5.804340666594078E-4	8.306638247084386E-4

From this data, I can conclude that the most optimal number of hidden nodes for 4 inputs is between 6 and 8.

No of Epochs

6 Hidden Nodes

```
Basic b = new Basic(4,6,39000,0.1);
b.run();
```

No of Epochs	Training MSE	Validation MSE
500	0.00066809	0.000895258
1000	0.000622103	0.000858907
1500	0.000600088	0.000843505
2000	0.000586965	0.000835266
2500	0.000577962	0.000830191
3000	0.000571541	0.000827115
3500	0.000566844	0.0008253
4000	0.000563206	0.00082405
4500	0.000560107	0.000822733
5000	0.000557158	0.000820801

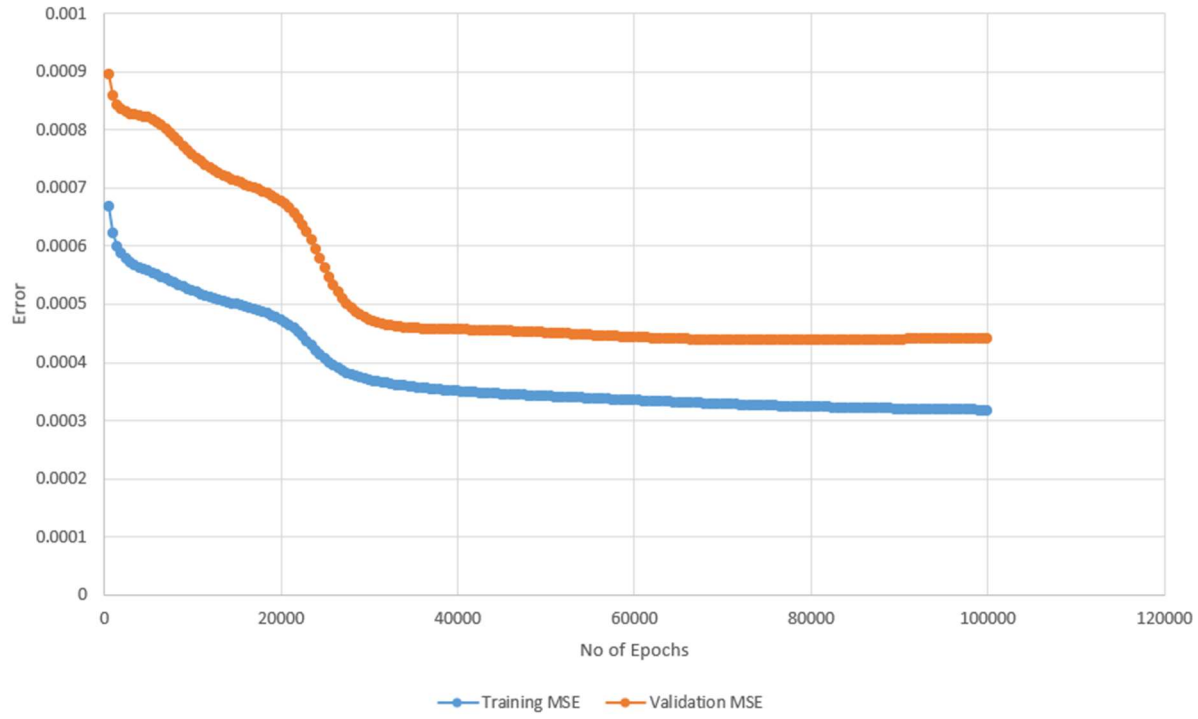
5500	0.000554103	0.000817835
6000	0.000550822	0.00081361
6500	0.000547334	0.00080815
7000	0.000543729	0.000801692
7500	0.000540106	0.000794565
8000	0.000536529	0.000787086
8500	0.000533037	0.000779512
9000	0.000529655	0.000772042
9500	0.000526398	0.000764819
10000	0.000523278	0.000757944
10500	0.000520304	0.000751478
11000	0.000517481	0.000745456
11500	0.000514811	0.000739886
12000	0.000512288	0.00073476
12500	0.000509906	0.000730056
13000	0.00050765	0.00072574
13500	0.000505504	0.000721774
14000	0.000503446	0.000718114
14500	0.000501451	0.000714712
15000	0.000499491	0.000711519
15500	0.000497535	0.000708482
16000	0.00049555	0.000705549
16500	0.000493497	0.000702661
17000	0.000491336	0.000699756
17500	0.000489022	0.000696764
18000	0.000486505	0.000693609
18500	0.000483731	0.000690198
19000	0.000480641	0.000686426
19500	0.000477171	0.000682167
20000	0.000473253	0.000677272
20500	0.000468815	0.000671565
21000	0.000463789	0.000664846
21500	0.000458114	0.000656894
22000	0.000451752	0.000647491
22500	0.000444712	0.000636465
23000	0.000437082	0.000623762
23500	0.000429056	0.000609526
24000	0.00042094	0.000594142
24500	0.000413096	0.0005782
25000	0.000405849	0.000562383
25500	0.000399407	0.000547322
26000	0.000393834	0.000533493
26500	0.000389079	0.000521177
27000	0.000385031	0.000510473

27500	0.000381568	0.000501341
28000	0.000378578	0.000493655
28500	0.000375968	0.000487241
29000	0.000373665	0.000481914
29500	0.000371609	0.000477499
30000	0.000369756	0.000473841
30500	0.000368069	0.000470806
31000	0.000366521	0.000468284
31500	0.000365091	0.000466187
32000	0.00036376	0.00046444
32500	0.000362517	0.000462984
33000	0.00036135	0.00046177
33500	0.000360252	0.000460759
34000	0.000359215	0.000459916
34500	0.000358235	0.000459215
35000	0.000357306	0.00045863
35500	0.000356424	0.000458144
36000	0.000355586	0.000457738
36500	0.000354788	0.000457397
37000	0.000354028	0.000457109
37500	0.000353302	0.000456864
38000	0.000352609	0.000456651
38500	0.000351945	0.000456461
39000	0.000351309	0.000456289
39500	0.000350699	0.000456127
40000	0.000350113	0.000455971
40500	0.000349548	0.000455816
41000	0.000349004	0.000455658
41500	0.000348479	0.000455493
42000	0.000347971	0.000455321
42500	0.00034748	0.000455137
43000	0.000347003	0.000454941
43500	0.00034654	0.000454732
44000	0.00034609	0.000454508
44500	0.000345652	0.000454269
45000	0.000345225	0.000454015
45500	0.000344807	0.000453746
46000	0.000344398	0.000453463
46500	0.000343997	0.000453164
47000	0.000343604	0.000452852
47500	0.000343217	0.000452527
48000	0.000342836	0.000452189
48500	0.00034246	0.00045184
49000	0.000342088	0.00045148

49500	0.00034172	0.000451111
50000	0.000341356	0.000450734
50500	0.000340994	0.000450349
51000	0.000340635	0.000449958
51500	0.000340278	0.000449562
52000	0.000339922	0.000449162
52500	0.000339568	0.000448758
53000	0.000339214	0.000448353
53500	0.000338862	0.000447946
54000	0.00033851	0.000447539
54500	0.000338158	0.000447133
55000	0.000337807	0.000446729
55500	0.000337457	0.000446328
56000	0.000337106	0.00044593
56500	0.000336756	0.000445536
57000	0.000336407	0.000445147
57500	0.000336058	0.000444765
58000	0.000335709	0.000444388
58500	0.000335362	0.000444019
59000	0.000335015	0.000443658
59500	0.000334669	0.000443306
60000	0.000334325	0.000442962
60500	0.000333981	0.000442628
61000	0.00033364	0.000442303
61500	0.0003333	0.000441989
62000	0.000332962	0.000441685
62500	0.000332627	0.000441393
63000	0.000332294	0.000441111
63500	0.000331963	0.000440841
64000	0.000331636	0.000440583
64500	0.000331311	0.000440336
65000	0.000330989	0.000440101
65500	0.000330671	0.000439877
66000	0.000330356	0.000439666
66500	0.000330044	0.000439466
67000	0.000329737	0.000439278
67500	0.000329433	0.000439101
68000	0.000329133	0.000438936
68500	0.000328837	0.000438782
69000	0.000328545	0.000438639
69500	0.000328257	0.000438508
70000	0.000327974	0.000438387
70500	0.000327695	0.000438277
71000	0.00032742	0.000438177

71500	0.000327149	0.000438087
72000	0.000326883	0.000438006
72500	0.000326621	0.000437936
73000	0.000326363	0.000437874
73500	0.00032611	0.000437822
74000	0.000325861	0.000437778
74500	0.000325616	0.000437743
75000	0.000325376	0.000437716
75500	0.00032514	0.000437697
76000	0.000324908	0.000437685
76500	0.00032468	0.000437681
77000	0.000324456	0.000437684
77500	0.000324237	0.000437693
78000	0.000324021	0.000437709
78500	0.00032381	0.000437731
79000	0.000323602	0.000437758
79500	0.000323398	0.000437792
80000	0.000323198	0.000437831
80500	0.000323002	0.000437875
81000	0.000322809	0.000437923
81500	0.00032262	0.000437976
82000	0.000322435	0.000438034
82500	0.000322253	0.000438096
83000	0.000322075	0.000438161
83500	0.0003219	0.00043823
84000	0.000321728	0.000438302
84500	0.000321559	0.000438378
85000	0.000321394	0.000438456
85500	0.000321231	0.000438537
86000	0.000321072	0.000438621
86500	0.000320915	0.000438706
87000	0.000320762	0.000438794
87500	0.000320611	0.000438884
88000	0.000320463	0.000438975
88500	0.000320317	0.000439068
89000	0.000320175	0.000439163
89500	0.000320034	0.000439258
90000	0.000319897	0.000439355
90500	0.000319761	0.000439453
91000	0.000319628	0.000439551
91500	0.000319498	0.00043965
92000	0.000319369	0.00043975
92500	0.000319243	0.00043985
93000	0.000319119	0.000439951

93500	0.000318997	0.000440051
94000	0.000318877	0.000440152
94500	0.000318758	0.000440253
95000	0.000318642	0.000440355
95500	0.000318528	0.000440456
96000	0.000318415	0.000440557
96500	0.000318305	0.000440657
97000	0.000318195	0.000440758
97500	0.000318088	0.000440858
98000	0.000317982	0.000440958
98500	0.000317878	0.000441058
99000	0.000317776	0.000441157
99500	0.000317675	0.000441256
100000	0.000317575	0.000441355



Overtraining occurs at 77,000 epochs

So I will train the data at 65,500 epochs:

Testing MSE: 4.376835375992998E-4

```
Weights:
[[0.3171095531981801, -0.08485668648407332, 0.49784405562528283, -2.6220497559090226, -0.2702739835258203, 2.8035340939226021
[-1.2682144352041267, 0.966672734274656, -2.879489803727609, 0.2363863184423585, 0.6907203249044724, 1.2233106737072794]
[-1.2644727709074182, -0.8642940200977761, -0.9956424924623501, -1.7066796873050938, -3.3333364069412874, 2.78168201176246]
[0.20917159983243014, 3.7012841551313445, -1.373712526288792, 0.9673744850177579, -1.8556333124168551, -4.143127427244809]
[-1.3512519094761308, -2.872971958171541, -2.5815689836711413, -2.3189033621960644, 3.4108560879774577, 5.958231839453706]]

Biases:
[0.08357940304160376, -3.7340163726853803, 1.3291164967831894, 1.158464491844384, 0.03086526939179807, -5.0887792021025895, 1.81150515051407021
```

Predicted Output	Actual Output

0.281264	0.27
0.631068	0.63
0.28677	0.3
0.66665	0.65
0.55639	0.55
0.547268	0.52
0.305795	0.3
0.583294	0.59
0.262766	0.29
0.641486	0.65
0.209529	0.2
0.785535	0.8
0.655061	0.66
0.733524	0.75
0.62903	0.65
0.520993	0.55
0.70851	0.73
0.62147	0.64
0.340678	0.33
0.494545	0.51
0.293116	0.31
0.38671	0.41
0.682317	0.69
0.617425	0.61
0.429335	0.45
0.167519	0.15
0.403443	0.43
0.555673	0.53
0.332457	0.32
0.59302	0.61
0.335233	0.36
0.26148	0.28
0.647889	0.64
0.493045	0.48
0.252169	0.26
0.540116	0.54
0.427182	0.42
0.515678	0.52
0.573942	0.59
0.545955	0.53
0.22803	0.23
0.268926	0.27
0.566433	0.55
0.712553	0.72

0.342937	0.35
0.567957	0.56
0.837123	0.82
0.214403	0.22
0.262542	0.24
0.764953	0.74
0.310348	0.29
0.512035	0.52
0.627693	0.62
0.67135	0.68
0.318182	0.36
0.557195	0.58
0.442495	0.46
0.313558	0.33
0.267314	0.26
0.473997	0.44
0.548435	0.56
0.287004	0.27
0.770623	0.77
0.428187	0.43
0.501244	0.52
0.48457	0.51
0.594383	0.62
0.357505	0.36
0.310917	0.29
0.75953	0.75
0.676113	0.7
0.589836	0.58
0.471029	0.5
0.68597	0.68
0.319579	0.35
0.348746	0.33
0.507141	0.5
0.304477	0.32
0.334519	0.34
0.329334	0.32
0.424856	0.37
0.471561	0.48
0.22957	0.21
0.37164	0.36
0.563777	0.56
0.241412	0.24
0.724719	0.71
0.229525	0.23

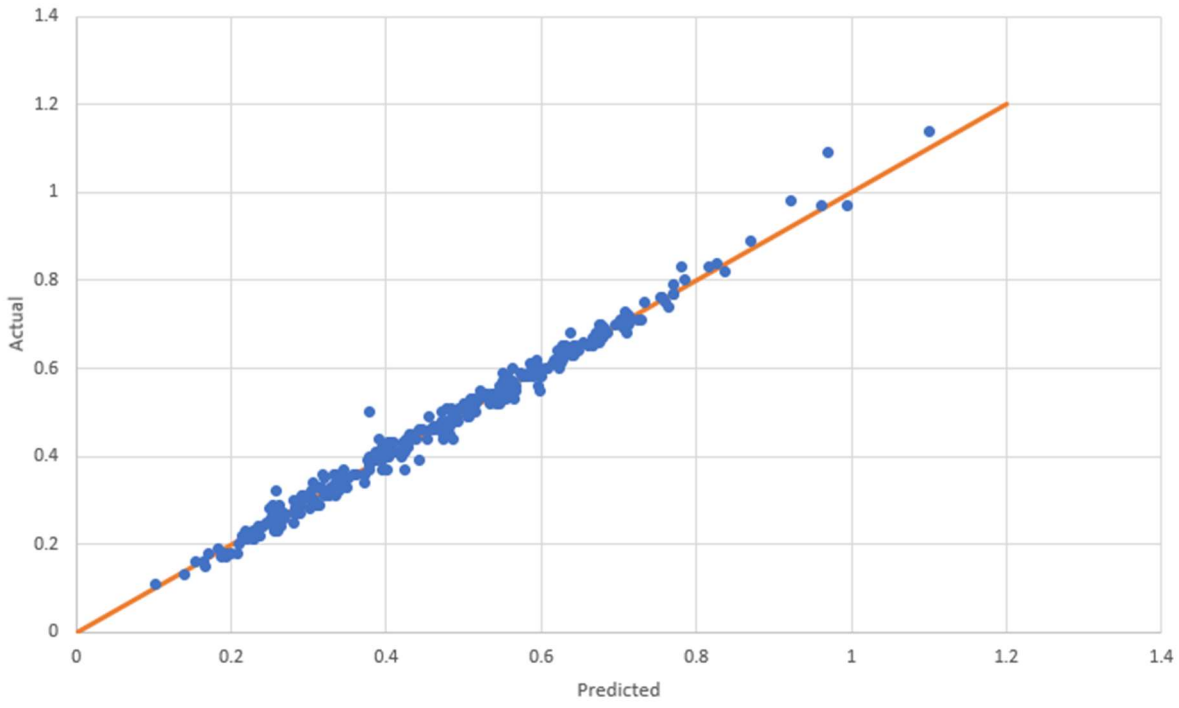
0.624304	0.61
0.961318	0.97
0.308124	0.3
0.709638	0.68
0.237069	0.22
0.438391	0.44
0.995299	0.97
0.597719	0.6
0.351904	0.35
0.754118	0.76
0.472886	0.45
0.164336	0.16
0.505383	0.49
0.409145	0.41
0.509214	0.53
0.826887	0.84
0.10179	0.11
0.482057	0.45
0.770363	0.79
0.28799	0.3
0.547242	0.56
0.266243	0.26
0.419953	0.4
0.485931	0.49
0.259416	0.23
0.675509	0.66
0.613979	0.61
0.401128	0.37
0.471281	0.46
0.520389	0.53
0.638741	0.64
0.378042	0.5
0.57575	0.58
0.25429	0.27
0.489521	0.49
0.618794	0.62
0.371865	0.36
0.712339	0.7
0.313185	0.29
0.711094	0.71
0.970097	1.09
0.637822	0.68
0.430981	0.44
0.345988	0.37

0.592022	0.6
0.516007	0.5
0.228021	0.22
0.404399	0.43
0.92214	0.98
0.289407	0.27
0.258666	0.28
0.528298	0.54
0.535455	0.53
0.729293	0.71
0.192727	0.18
0.386916	0.4
0.208679	0.18
0.192604	0.17
0.485167	0.5
0.770561	0.77
0.30517	0.34
1.100782	1.14
0.494714	0.51
0.550678	0.59
0.626803	0.65
0.337863	0.34
0.585403	0.61
0.536172	0.54
0.403044	0.4
0.254046	0.29
0.344897	0.35
0.442756	0.39
0.425799	0.44
0.678449	0.67
0.50433	0.5
0.303684	0.3
0.382261	0.39
0.384128	0.4
0.386914	0.39
0.2488	0.28
0.756364	0.76
0.220823	0.21
0.467692	0.46
0.289886	0.29
0.57958	0.58
0.28286	0.28
0.597411	0.59
0.599852	0.58

0.461346	0.46
0.454525	0.49
0.39565	0.39
0.453721	0.44
0.409584	0.43
0.234357	0.24
0.563217	0.56
0.513433	0.53
0.325987	0.31
0.314971	0.33
0.402738	0.43
0.424315	0.43
0.182972	0.19
0.478712	0.48
0.486962	0.44
0.139451	0.13
0.622659	0.6
0.676389	0.7
0.339185	0.32
0.188312	0.17
0.551538	0.57
0.599276	0.6
0.170249	0.18
0.661244	0.65
0.481996	0.47
0.665642	0.66
0.529914	0.54
0.66975	0.67
0.384302	0.4
0.395644	0.37
0.641674	0.63
0.534511	0.52
0.390585	0.44
0.42207	0.42
0.24513	0.25
0.258702	0.32
0.562921	0.55
0.422929	0.41
0.285181	0.27
0.616728	0.62
0.291953	0.31
0.705765	0.69
0.379039	0.37
0.479518	0.46

0.643038	0.65
0.638952	0.63
0.268724	0.26
0.199255	0.18
0.332464	0.36
0.284438	0.27
0.283499	0.28
0.478638	0.51
0.489772	0.48
0.816219	0.83
0.507504	0.51
0.56387	0.57
0.71732	0.71
0.607522	0.6
0.301208	0.28
0.422078	0.43
0.395888	0.42
0.501484	0.52
0.259855	0.27
0.371925	0.34
0.228255	0.21
0.462253	0.47
0.595497	0.56
0.361423	0.36
0.44762	0.46
0.626465	0.64
0.375602	0.39
0.609216	0.6
0.395467	0.4
0.644032	0.64
0.623843	0.63
0.254763	0.24
0.2844	0.28
0.546177	0.55
0.543001	0.52
0.507022	0.49
0.401784	0.41
0.280809	0.25
0.870817	0.89
0.255229	0.23
0.264792	0.24
0.281521	0.3
0.565396	0.53
0.219391	0.23

0.153409	0.16
0.598417	0.55
0.562887	0.6
0.474929	0.47
0.436834	0.45
0.518912	0.53
0.377587	0.38
0.377685	0.4
0.41171	0.41
0.531951	0.53
0.584185	0.58
0.388951	0.39
0.333736	0.31
0.696001	0.7
0.702355	0.71
0.457045	0.46
0.291105	0.3
0.526238	0.54
0.594723	0.59
0.780001	0.83
0.26124	0.25
0.319573	0.31
0.63152	0.65
0.667277	0.67
0.329106	0.33
0.51261	0.51



8 Hidden Nodes

```
Basic b = new Basic(4,8,39000,0.1);
b.run();
```

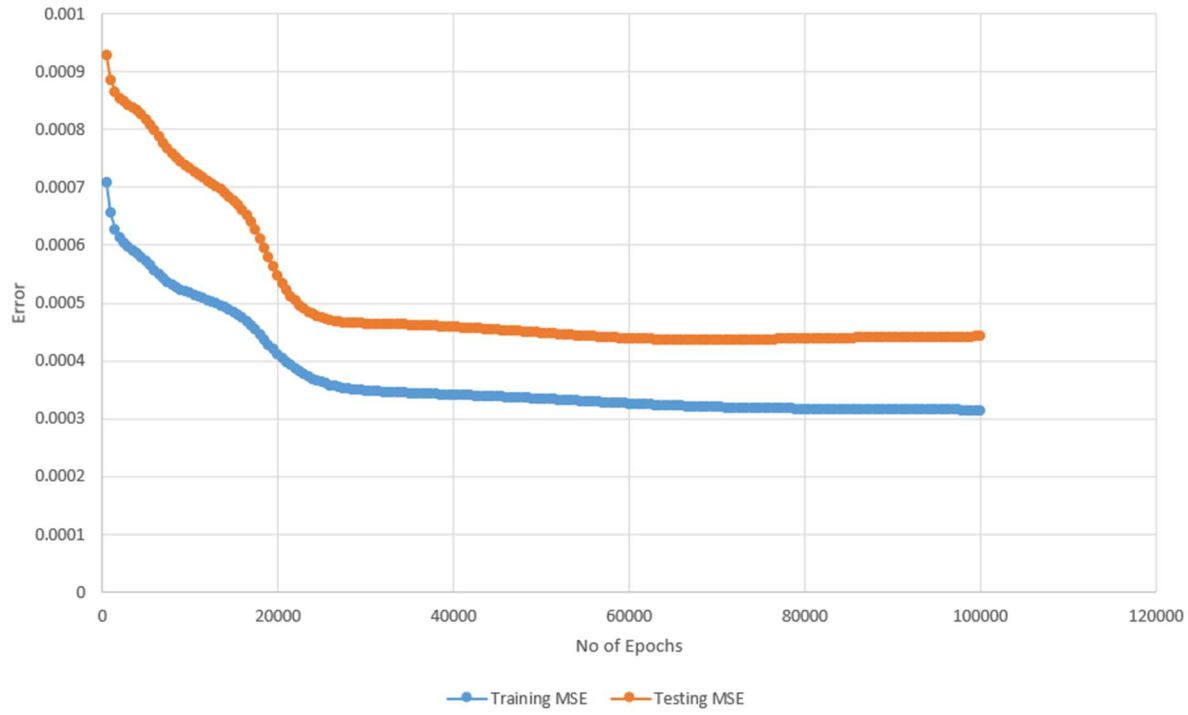
No of Epochs	Training MSE	Testing MSE
500	0.000707582	0.000927884
1000	0.000655021	0.000885484
1500	0.000627595	0.000864692
2000	0.000612677	0.000854157
2500	0.00060364	0.000847822
3000	0.000596976	0.000842658
3500	0.000591084	0.000837438
4000	0.000585267	0.000831704
4500	0.000579158	0.000825177
5000	0.000572473	0.000817502
5500	0.000565042	0.000808326
6000	0.000557055	0.000797718
6500	0.000549171	0.000786495
7000	0.0005421	0.000775773
7500	0.000536136	0.000766221
8000	0.000531164	0.00075791
8500	0.000526928	0.000750609
9000	0.000523193	0.000744042
9500	0.000519791	0.000737989
10000	0.000516612	0.000732299
10500	0.000513581	0.000726872
11000	0.000510649	0.000721639
11500	0.000507771	0.000716537
12000	0.000504907	0.000711505
12500	0.000502009	0.00070647
13000	0.000499024	0.000701344
13500	0.000495888	0.000696021
14000	0.000492524	0.000690377
14500	0.000488846	0.000684259
15000	0.000484753	0.000677488
15500	0.000480133	0.00066986
16000	0.000474869	0.000661143
16500	0.00046885	0.000651098
17000	0.000461998	0.000639518
17500	0.000454303	0.000626289
18000	0.000445874	0.000611486
18500	0.000436972	0.000595464
19000	0.000427987	0.000578874

19500	0.000419341	0.000562548
20000	0.000411353	0.000547274
20500	0.000404152	0.000533586
21000	0.000397699	0.000521691
21500	0.00039187	0.000511544
22000	0.000386543	0.000502964
22500	0.000381634	0.000495738
23000	0.000377106	0.000489671
23500	0.00037295	0.000484595
24000	0.000369171	0.000480375
24500	0.000365776	0.000476896
25000	0.000362764	0.000474055
25500	0.000360124	0.000471761
26000	0.000357835	0.000469931
26500	0.000355868	0.000468486
27000	0.000354188	0.000467356
27500	0.000352757	0.000466477
28000	0.000351539	0.000465795
28500	0.000350499	0.000465264
29000	0.000349607	0.000464846
29500	0.000348836	0.000464509
30000	0.000348163	0.000464229
30500	0.000347569	0.000463987
31000	0.000347039	0.000463767
31500	0.00034656	0.000463558
32000	0.000346122	0.000463352
32500	0.000345717	0.000463142
33000	0.000345337	0.000462924
33500	0.000344979	0.000462694
34000	0.000344638	0.000462451
34500	0.000344309	0.000462192
35000	0.000343992	0.000461916
35500	0.000343682	0.000461624
36000	0.000343379	0.000461316
36500	0.000343081	0.00046099
37000	0.000342786	0.000460649
37500	0.000342493	0.000460292
38000	0.000342202	0.00045992
38500	0.000341911	0.000459533
39000	0.00034162	0.000459132
39500	0.000341328	0.000458718
40000	0.000341034	0.000458292
40500	0.000340738	0.000457854
41000	0.000340439	0.000457405

41500	0.000340137	0.000456945
42000	0.000339832	0.000456475
42500	0.000339522	0.000455996
43000	0.000339208	0.000455508
43500	0.00033889	0.000455012
44000	0.000338566	0.000454508
44500	0.000338238	0.000453998
45000	0.000337904	0.000453481
45500	0.000337564	0.000452958
46000	0.000337218	0.000452429
46500	0.000336866	0.000451896
47000	0.000336508	0.000451359
47500	0.000336144	0.000450818
48000	0.000335773	0.000450275
48500	0.000335397	0.00044973
49000	0.000335014	0.000449183
49500	0.000334625	0.000448635
50000	0.000334231	0.000448088
50500	0.00033383	0.000447542
51000	0.000333425	0.000446998
51500	0.000333014	0.000446457
52000	0.000332599	0.00044592
52500	0.00033218	0.000445388
53000	0.000331757	0.000444862
53500	0.000331331	0.000444342
54000	0.000330902	0.000443831
54500	0.000330471	0.000443329
55000	0.000330039	0.000442837
55500	0.000329607	0.000442356
56000	0.000329175	0.000441887
56500	0.000328743	0.000441431
57000	0.000328314	0.00044099
57500	0.000327887	0.000440563
58000	0.000327463	0.000440153
58500	0.000327043	0.000439759
59000	0.000326628	0.000439384
59500	0.000326219	0.000439026
60000	0.000325815	0.000438687
60500	0.000325419	0.000438368
61000	0.000325029	0.000438068
61500	0.000324648	0.000437789
62000	0.000324275	0.00043753
62500	0.000323912	0.000437291
63000	0.000323557	0.000437073

63500	0.000323212	0.000436876
64000	0.000322877	0.000436699
64500	0.000322553	0.000436542
65000	0.000322238	0.000436405
65500	0.000321934	0.000436288
66000	0.00032164	0.000436189
66500	0.000321357	0.000436109
67000	0.000321084	0.000436047
67500	0.000320821	0.000436002
68000	0.000320568	0.000435973
68500	0.000320325	0.000435959
69000	0.000320092	0.00043596
69500	0.000319869	0.000435975
70000	0.000319654	0.000436004
70500	0.000319449	0.000436044
71000	0.000319252	0.000436095
71500	0.000319063	0.000436157
72000	0.000318882	0.000436228
72500	0.000318709	0.000436308
73000	0.000318543	0.000436396
73500	0.000318384	0.00043649
74000	0.000318232	0.000436591
74500	0.000318087	0.000436697
75000	0.000317947	0.000436808
75500	0.000317813	0.000436923
76000	0.000317684	0.000437041
76500	0.000317561	0.000437162
77000	0.000317443	0.000437285
77500	0.000317329	0.00043741
78000	0.00031722	0.000437536
78500	0.000317115	0.000437664
79000	0.000317014	0.000437791
79500	0.000316916	0.000437919
80000	0.000316822	0.000438047
80500	0.000316732	0.000438174
81000	0.000316644	0.0004383
81500	0.00031656	0.000438426
82000	0.000316478	0.000438551
82500	0.0003164	0.000438674
83000	0.000316323	0.000438796
83500	0.000316249	0.000438916
84000	0.000316178	0.000439035
84500	0.000316108	0.000439152
85000	0.000316041	0.000439268

85500	0.000315975	0.000439381
86000	0.000315912	0.000439493
86500	0.00031585	0.000439603
87000	0.00031579	0.00043971
87500	0.000315731	0.000439816
88000	0.000315674	0.00043992
88500	0.000315619	0.000440022
89000	0.000315565	0.000440121
89500	0.000315512	0.000440219
90000	0.00031546	0.000440315
90500	0.00031541	0.000440408
91000	0.000315361	0.0004405
91500	0.000315313	0.00044059
92000	0.000315266	0.000440677
92500	0.00031522	0.000440763
93000	0.000315175	0.000440847
93500	0.000315131	0.000440928
94000	0.000315088	0.000441008
94500	0.000315046	0.000441086
95000	0.000315005	0.000441162
95500	0.000314965	0.000441236
96000	0.000314925	0.000441309
96500	0.000314886	0.000441379
97000	0.000314848	0.000441448
97500	0.00031481	0.000441515
98000	0.000314774	0.00044158
98500	0.000314737	0.000441643
99000	0.000314702	0.000441705
99500	0.000314667	0.000441765
100000	0.000314633	0.000441824



Overtraining occurs at 69,000 epochs

So I will train the data at 68,500 epochs

Testing MSE: 4.35959071181917E-4

Weights:
[[-0.11406114035219093, 3.1876755401444044, -0.558362630852506, -2.9694873752654667, -0.57048304893101 48, 1.6697265491010693, 0.38629785368790 99, 0.08878871610118 47]
[0.6229189290893224, 1.3109139584019043, -0.11640500533835602, 0.496744529678134, 0.4424538514633998, -0.7675914399348123, -3.021312414287736, -0.14662003312116592]
[-2.718415988398845, 3.1453752442161096, -1.1309659591343595, -1.8402675823406607, -0.2909964778343635, -1.5396045568488734, -0.6917254462713711, 0.18581789111983918]
[-2.1552369816748675, -4.540009043370342, 0.6082032540724447, 0.8813963249720621, 3.3293320547418497, 1.4366581919266 47, -2.7590214553560983, -0.5682727536655306]
[3.452355737202362, 5.411433677677219, -0.8976478959456919, -2.243177253931928, -2.528473577180549, -1.7812628214010113, -2.6215469407805476, 0.9084785282192695]]

Biases:
[0.24667665972227298, -5.507025487119143, 0.4600764367770932, 1.4451029567856957, -3.165806456984994, -1.4269444802848912, 1.9877380831211615, -0.12026641062697217, 1.895693548067523]

Predicted Output	Actual Output
0.28188	0.27
0.630761	0.63
0.287088	0.3
0.665751	0.65
0.555747	0.55
0.547074	0.52
0.302328	0.3
0.582804	0.59
0.262723	0.29
0.64105	0.65
0.212149	0.2
0.787394	0.8
0.654485	0.66
0.735073	0.75
0.626754	0.65
0.521302	0.55

0.712682	0.73
0.622261	0.64
0.340553	0.33
0.49672	0.51
0.290722	0.31
0.387282	0.41
0.682912	0.69
0.616695	0.61
0.429439	0.45
0.170074	0.15
0.403373	0.43
0.555487	0.53
0.332895	0.32
0.593163	0.61
0.335142	0.36
0.262286	0.28
0.648063	0.64
0.494008	0.48
0.251963	0.26
0.540697	0.54
0.427088	0.42
0.51549	0.52
0.574734	0.59
0.546214	0.53
0.228213	0.23
0.269467	0.27
0.566297	0.55
0.713672	0.72
0.343806	0.35
0.567015	0.56
0.839229	0.82
0.213691	0.22
0.26194	0.24
0.766651	0.74
0.310834	0.29
0.51105	0.52
0.627129	0.62
0.672048	0.68
0.317797	0.36
0.5578	0.58
0.439695	0.46
0.31158	0.33
0.267337	0.26
0.476753	0.44

0.54841	0.56
0.287358	0.27
0.772625	0.77
0.426581	0.43
0.500533	0.52
0.483255	0.51
0.595153	0.62
0.357615	0.36
0.311542	0.29
0.760564	0.75
0.676266	0.7
0.590048	0.58
0.467837	0.5
0.685896	0.68
0.319488	0.35
0.349161	0.33
0.508289	0.5
0.304885	0.32
0.334915	0.34
0.330176	0.32
0.427135	0.37
0.470892	0.48
0.23	0.21
0.371592	0.36
0.563283	0.56
0.243654	0.24
0.725561	0.71
0.228757	0.23
0.624219	0.61
0.960244	0.97
0.308626	0.3
0.710566	0.68
0.237144	0.22
0.438751	0.44
0.996988	0.97
0.597553	0.6
0.352056	0.35
0.754681	0.76
0.476264	0.45
0.164054	0.16
0.50487	0.49
0.405814	0.41
0.509009	0.53
0.828934	0.84

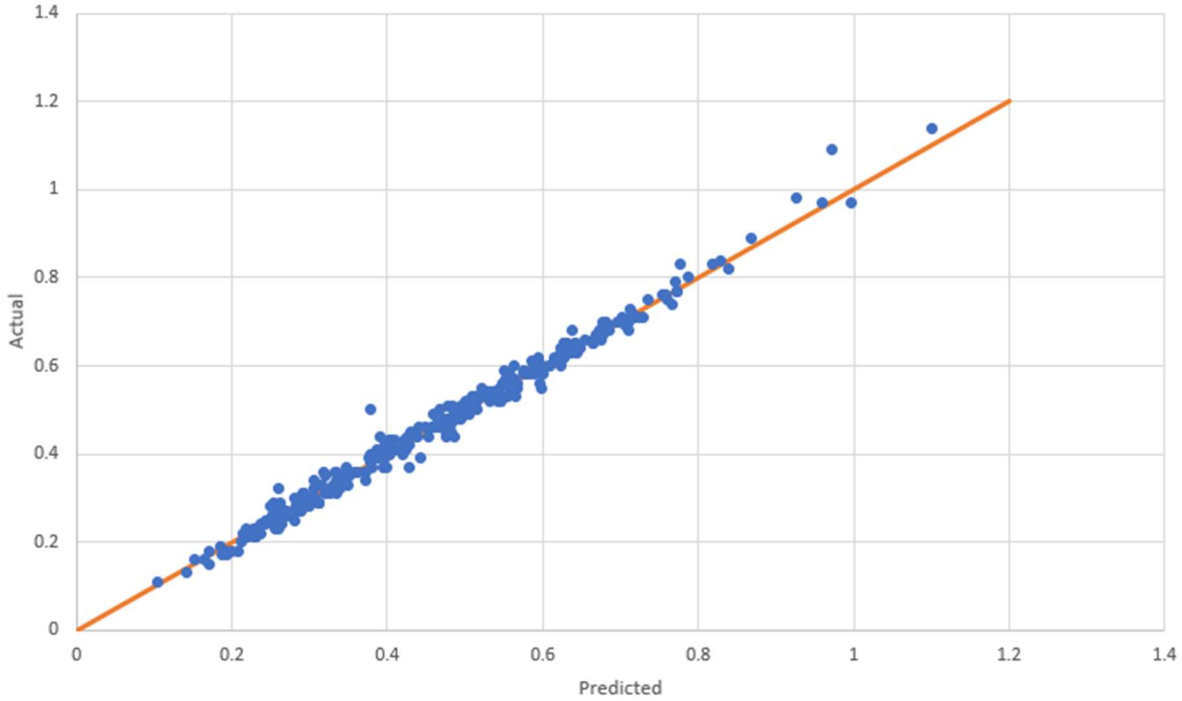
0.103427	0.11
0.482123	0.45
0.771153	0.79
0.28865	0.3
0.548008	0.56
0.266254	0.26
0.41984	0.4
0.485737	0.49
0.259589	0.23
0.675769	0.66
0.614219	0.61
0.399282	0.37
0.470869	0.46
0.524422	0.53
0.638859	0.64
0.378456	0.5
0.575397	0.58
0.25493	0.27
0.489323	0.49
0.619493	0.62
0.370371	0.36
0.70962	0.7
0.312789	0.29
0.711835	0.71
0.971555	1.09
0.638748	0.68
0.430339	0.44
0.346275	0.37
0.591976	0.6
0.515205	0.5
0.227685	0.22
0.404533	0.43
0.92544	0.98
0.288063	0.27
0.259553	0.28
0.528576	0.54
0.534543	0.53
0.728757	0.71
0.193319	0.18
0.386875	0.4
0.20909	0.18
0.192758	0.17
0.484104	0.5
0.772082	0.77

0.305623	0.34
1.099808	1.14
0.493934	0.51
0.550442	0.59
0.627371	0.65
0.338344	0.34
0.586244	0.61
0.535648	0.54
0.403512	0.4
0.253994	0.29
0.3449	0.35
0.443074	0.39
0.426068	0.44
0.67797	0.67
0.504191	0.5
0.30433	0.3
0.382546	0.39
0.384179	0.4
0.386829	0.39
0.24922	0.28
0.75782	0.76
0.22054	0.21
0.467354	0.46
0.290325	0.29
0.579501	0.58
0.283509	0.28
0.597822	0.59
0.600164	0.58
0.46228	0.46
0.459799	0.49
0.395939	0.39
0.453005	0.44
0.409701	0.43
0.236326	0.24
0.562097	0.56
0.513825	0.53
0.326037	0.31
0.315689	0.33
0.402734	0.43
0.424031	0.43
0.184359	0.19
0.478731	0.48
0.486853	0.44
0.142555	0.13

0.623618	0.6
0.682286	0.7
0.338316	0.32
0.187704	0.17
0.551818	0.57
0.599209	0.6
0.170269	0.18
0.664738	0.65
0.481663	0.47
0.665553	0.66
0.529128	0.54
0.669676	0.67
0.384038	0.4
0.395582	0.37
0.64299	0.63
0.53231	0.52
0.391321	0.44
0.422081	0.42
0.244364	0.25
0.259405	0.32
0.562063	0.55
0.423122	0.41
0.284647	0.27
0.615558	0.62
0.292631	0.31
0.706039	0.69
0.379486	0.37
0.479422	0.46
0.6423	0.65
0.638364	0.63
0.269617	0.26
0.198987	0.18
0.332541	0.36
0.284883	0.27
0.283201	0.28
0.478376	0.51
0.488687	0.48
0.818126	0.83
0.50689	0.51
0.563378	0.57
0.718229	0.71
0.607675	0.6
0.299694	0.28
0.421183	0.43

0.396262	0.42
0.502424	0.52
0.261272	0.27
0.372276	0.34
0.228173	0.21
0.462345	0.47
0.595409	0.56
0.361506	0.36
0.447896	0.46
0.627085	0.64
0.375741	0.39
0.6095	0.6
0.395369	0.4
0.643951	0.64
0.623821	0.63
0.254064	0.24
0.284432	0.28
0.544943	0.55
0.542488	0.52
0.50537	0.49
0.401862	0.41
0.281454	0.25
0.868558	0.89
0.255337	0.23
0.265001	0.24
0.28071	0.3
0.565356	0.53
0.219456	0.23
0.152871	0.16
0.599304	0.55
0.562757	0.6
0.474386	0.47
0.437138	0.45
0.519224	0.53
0.377343	0.38
0.37782	0.4
0.411059	0.41
0.531646	0.53
0.584478	0.58
0.388887	0.39
0.333876	0.31
0.69666	0.7
0.70259	0.71
0.457147	0.46

0.291522	0.3
0.527269	0.54
0.593602	0.59
0.777244	0.83
0.261906	0.25
0.319821	0.31
0.632168	0.65
0.667938	0.67
0.329735	0.33
0.512685	0.51



Decision

No of Hidden Nodes	Testing MSE minima	No of Epochs
6	4.376835375992998E-4	75,000
8	4.35959071181917E-4	68,500

It is clear that 8 hidden nodes it the most optimal of the two

Final Decision

No of Inputs	No of Hidden Nodes	Testing MSE minima	No of Epochs
5	4	4.493479551351621E-4	39,000
4	8	4.35959071181917E-4	68,500

For 4 inputs, it took much longer to train the data. However, the final output is more accurate in terms of predicting unseen data

I will therefore use this against the Model modification in an attempt to reduce the no of epochs it will take to reach this minima.

Evaluation of final Model

Now I have found the optimal number of hidden nodes, I will use this, along with some additional modifications to the Basic Model, to find out our final model that gives the lowest Mean Squared Error for our Testing and Validation Datasets.

I have implemented all the following Modifications in their own separate class:

- Momentum
- Bold Driver
- Annealing
- Weight Decay

And will go through each of these with a hidden node of 3 and find out which one gives the lowest mean squared error.

Momentum

```
Momentum m = new Momentum(4,8,100000,0.1,0.9);
m.run();
```

No of Epochs	Training MSE	Testing MSE
500	0.00061155	0.00084064
1000	0.000545525	0.000738493
1500	0.000458956	0.00060822
2000	0.000409924	0.000510705
2500	0.000392054	0.000480258
3000	0.000376683	0.000462415
3500	0.000360731	0.000450998
4000	0.000348804	0.000446731
4500	0.000342184	0.000446415
5000	0.000338336	0.00044662
5500	0.000335628	0.000446357
6000	0.000333461	0.000445792
6500	0.000331629	0.000445118
7000	0.000330015	0.000444358
7500	0.00032853	0.000443545
8000	0.000327116	0.000442782
8500	0.000325753	0.000442183
9000	0.000324444	0.000441825
9500	0.000323208	0.000441734
10000	0.000322067	0.000441895
10500	0.00032104	0.000442268
11000	0.00032014	0.000442802

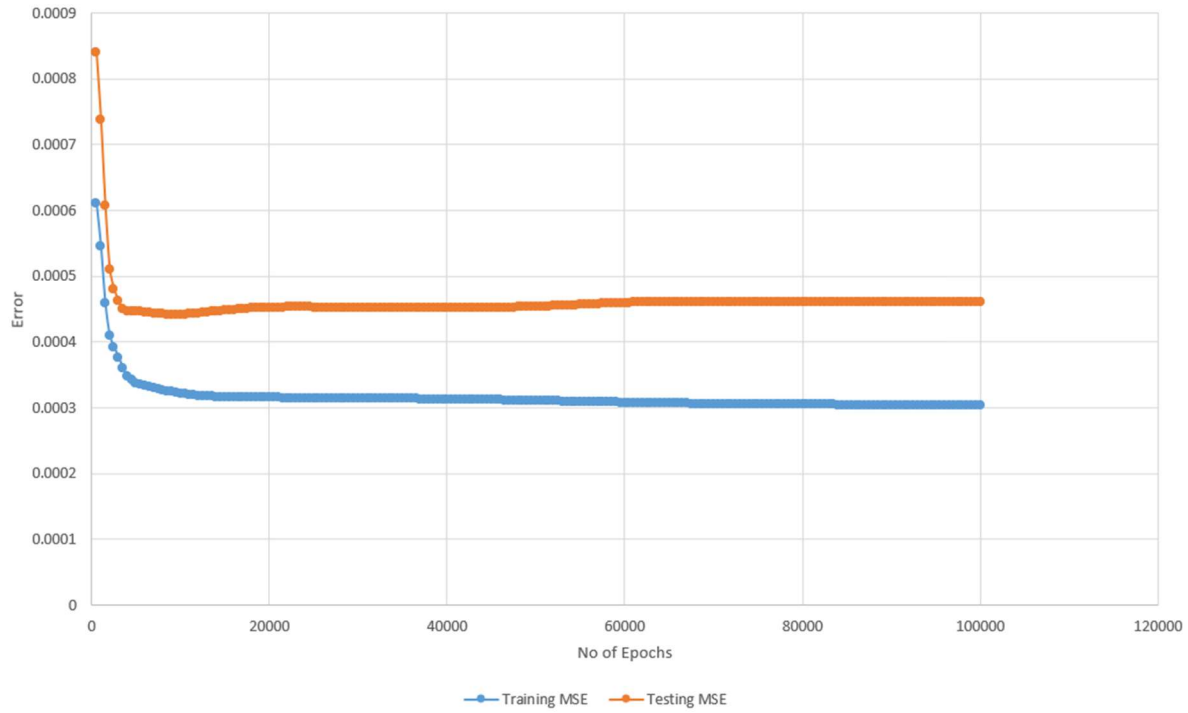
11500	0.000319371	0.000443448
12000	0.000318726	0.000444162
12500	0.000318194	0.00044491
13000	0.000317757	0.000445665
13500	0.0003174	0.000446408
14000	0.000317109	0.000447125
14500	0.000316869	0.000447809
15000	0.000316671	0.000448453
15500	0.000316506	0.000449055
16000	0.000316367	0.000449613
16500	0.000316248	0.000450129
17000	0.000316145	0.000450602
17500	0.000316055	0.000451034
18000	0.000315975	0.000451426
18500	0.000315903	0.00045178
19000	0.000315836	0.000452098
19500	0.000315773	0.00045238
20000	0.000315713	0.000452627
20500	0.000315654	0.00045284
21000	0.000315595	0.000453018
21500	0.000315535	0.000453161
22000	0.000315473	0.000453269
22500	0.000315408	0.000453341
23000	0.00031534	0.000453375
23500	0.000315268	0.000453372
24000	0.000315192	0.000453332
24500	0.000315113	0.000453258
25000	0.000315032	0.000453153
25500	0.00031495	0.000453026
26000	0.000314869	0.000452887
26500	0.000314793	0.000452746
27000	0.000314722	0.000452614
27500	0.000314658	0.0004525
28000	0.000314602	0.00045241
28500	0.000314554	0.000452347
29000	0.000314511	0.000452308
29500	0.000314473	0.000452292
30000	0.000314437	0.000452291
30500	0.000314403	0.000452302
31000	0.000314369	0.00045232
31500	0.000314333	0.000452339
32000	0.000314295	0.000452358
32500	0.000314255	0.000452373
33000	0.000314211	0.000452385

33500	0.000314165	0.000452392
34000	0.000314116	0.000452394
34500	0.000314065	0.000452392
35000	0.00031401	0.000452386
35500	0.000313953	0.000452377
36000	0.000313893	0.000452366
36500	0.000313831	0.000452354
37000	0.000313767	0.00045234
37500	0.0003137	0.000452326
38000	0.00031363	0.000452312
38500	0.000313558	0.0004523
39000	0.000313483	0.000452288
39500	0.000313406	0.000452279
40000	0.000313325	0.000452272
40500	0.000313242	0.000452268
41000	0.000313155	0.000452268
41500	0.000313064	0.000452272
42000	0.00031297	0.000452282
42500	0.000312873	0.000452299
43000	0.000312771	0.000452322
43500	0.000312666	0.000452354
44000	0.000312557	0.000452396
44500	0.000312445	0.000452448
45000	0.000312329	0.000452512
45500	0.00031221	0.00045259
46000	0.000312088	0.000452682
46500	0.000311962	0.000452789
47000	0.000311835	0.000452913
47500	0.000311705	0.000453054
48000	0.000311573	0.000453212
48500	0.000311439	0.000453387
49000	0.000311304	0.00045358
49500	0.000311167	0.00045379
50000	0.000311029	0.000454017
50500	0.000310889	0.00045426
51000	0.000310749	0.000454517
51500	0.000310608	0.000454788
52000	0.000310466	0.000455071
52500	0.000310324	0.000455365
53000	0.000310182	0.000455668
53500	0.000310039	0.000455977
54000	0.000309897	0.000456292
54500	0.000309755	0.000456611
55000	0.000309614	0.00045693

55500	0.000309474	0.00045725
56000	0.000309334	0.000457567
56500	0.000309196	0.00045788
57000	0.00030906	0.000458187
57500	0.000308925	0.000458487
58000	0.000308791	0.000458777
58500	0.00030866	0.000459058
59000	0.000308531	0.000459327
59500	0.000308404	0.000459583
60000	0.000308279	0.000459826
60500	0.000308156	0.000460055
61000	0.000308036	0.000460269
61500	0.000307918	0.000460468
62000	0.000307803	0.000460652
62500	0.00030769	0.000460821
63000	0.00030758	0.000460974
63500	0.000307472	0.000461113
64000	0.000307366	0.000461237
64500	0.000307263	0.000461347
65000	0.000307162	0.000461444
65500	0.000307064	0.000461527
66000	0.000306969	0.000461599
66500	0.000306875	0.000461659
67000	0.000306785	0.000461709
67500	0.000306696	0.000461749
68000	0.00030661	0.00046178
68500	0.000306527	0.000461803
69000	0.000306446	0.000461819
69500	0.000306367	0.000461828
70000	0.000306291	0.000461832
70500	0.000306217	0.00046183
71000	0.000306146	0.000461825
71500	0.000306077	0.000461815
72000	0.00030601	0.000461803
72500	0.000305946	0.000461788
73000	0.000305884	0.000461771
73500	0.000305824	0.000461752
74000	0.000305766	0.000461732
74500	0.00030571	0.000461711
75000	0.000305657	0.000461689
75500	0.000305605	0.000461668
76000	0.000305555	0.000461645
76500	0.000305507	0.000461623
77000	0.00030546	0.000461602

77500	0.000305416	0.00046158
78000	0.000305372	0.00046156
78500	0.000305331	0.00046154
79000	0.000305291	0.00046152
79500	0.000305252	0.000461502
80000	0.000305214	0.000461484
80500	0.000305178	0.000461468
81000	0.000305142	0.000461452
81500	0.000305108	0.000461438
82000	0.000305075	0.000461425
82500	0.000305042	0.000461413
83000	0.000305011	0.000461402
83500	0.00030498	0.000461393
84000	0.00030495	0.000461384
84500	0.00030492	0.000461378
85000	0.000304892	0.000461372
85500	0.000304864	0.000461368
86000	0.000304836	0.000461365
86500	0.000304809	0.000461364
87000	0.000304782	0.000461364
87500	0.000304756	0.000461366
88000	0.00030473	0.000461369
88500	0.000304704	0.000461374
89000	0.000304679	0.00046138
89500	0.000304654	0.000461388
90000	0.00030463	0.000461398
90500	0.000304606	0.000461409
91000	0.000304582	0.000461421
91500	0.000304558	0.000461436
92000	0.000304535	0.000461452
92500	0.000304511	0.000461469
93000	0.000304489	0.000461489
93500	0.000304466	0.00046151
94000	0.000304444	0.000461532
94500	0.000304421	0.000461557
95000	0.0003044	0.000461583
95500	0.000304378	0.00046161
96000	0.000304356	0.00046164
96500	0.000304335	0.000461671
97000	0.000304314	0.000461704
97500	0.000304294	0.000461738
98000	0.000304274	0.000461774
98500	0.000304253	0.000461812
99000	0.000304234	0.000461851

99500	0.000304214	0.000461892
100000	0.000304195	0.000461934



Overtraining Starts to occur at around 10,000 epochs, which is much sooner than when using the basic model

So I will run the algorithm at 9500 epochs and investigate the data:

```
Momentum m = new Momentum(4,8,9500,0.1,0.9);
m.run();
```

Testing MSE: 4.4173397378744655E-4

```
Weights:
[[2.8392178052092185, -0.9464478217579466, 0.633070760673127, -1.0015321004543154, -0.17186157981153485, 0.8211109060721535, -0.0765017496747835, -3.06241286161295671
[-0.3572959725600063, 0.0103023990783566O4, -3.8300946728526553, -0.5502777207481B4, -1.0505716660524247, 0.14738798310202555, 0.8343672060560007, -1.627713789856296]
[2.471225096401913, -0.555851622456O479, -0.8439748999285162, -0.6423888048652279, -1.5827354623774093, 1.9007390206177779, -2.621985684806938, -3.796493347127306411
[-1.5855095241388721, -0.067616502250434O9, -4.263820211004867, 0.10093529660264525, 4.45033390210282, -1.8049552340108255, -2.9715121389649712, 5.037354638593269511
[1.6564327579418658, -0.18675551505371746, -1.7601552111511343, -0.13807788935106477, -1.840553390060843, 1.4243634348839347, 2.8069902030466514, -4.8741279843518181]]

Biases:
[-1.3320622058407838, -1.3668536034767822, 2.831922983681649S, -1.183337460629492, -2.5004625895202937, -2.140401104133136, 0.48256165344297863, 5.763521527174522, 3.80294720883840131
```

Predicted Value	Actual Value
0.280805	0.27
0.628206	0.63
0.285299	0.3
0.670427	0.65
0.554414	0.55
0.54389	0.52
0.299513	0.3
0.582149	0.59

0.260455	0.29
0.640809	0.65
0.21397	0.2
0.783578	0.8
0.654071	0.66
0.733924	0.75
0.624125	0.65
0.522655	0.55
0.710675	0.73
0.621416	0.64
0.34209	0.33
0.501652	0.51
0.287292	0.31
0.38728	0.41
0.68255	0.69
0.61442	0.61
0.428959	0.45
0.172716	0.15
0.404184	0.43
0.553682	0.53
0.333805	0.32
0.59374	0.61
0.333077	0.36
0.260747	0.28
0.646238	0.64
0.495627	0.48
0.25154	0.26
0.541891	0.54
0.426381	0.42
0.514553	0.52
0.573946	0.59
0.546793	0.53
0.226861	0.23
0.267903	0.27
0.56765	0.55
0.712101	0.72
0.344135	0.35
0.56463	0.56
0.837501	0.82
0.21174	0.22
0.259531	0.24
0.765864	0.74
0.308972	0.29
0.508962	0.52

0.62411	0.62
0.669811	0.68
0.317815	0.36
0.559303	0.58
0.44754	0.46
0.313765	0.33
0.266637	0.26
0.478361	0.44
0.547565	0.56
0.287678	0.27
0.769147	0.77
0.424717	0.43
0.506678	0.52
0.481491	0.51
0.596206	0.62
0.356519	0.36
0.31201	0.29
0.758809	0.75
0.675283	0.7
0.590852	0.58
0.473449	0.5
0.682266	0.68
0.318214	0.35
0.347516	0.33
0.509577	0.5
0.303477	0.32
0.335091	0.34
0.331029	0.32
0.426203	0.37
0.47173	0.48
0.228909	0.21
0.371043	0.36
0.560799	0.56
0.2441	0.24
0.727161	0.71
0.226217	0.23
0.622743	0.61
0.96711	0.97
0.308932	0.3
0.709323	0.68
0.235821	0.22
0.43959	0.44
0.993242	0.97
0.596461	0.6

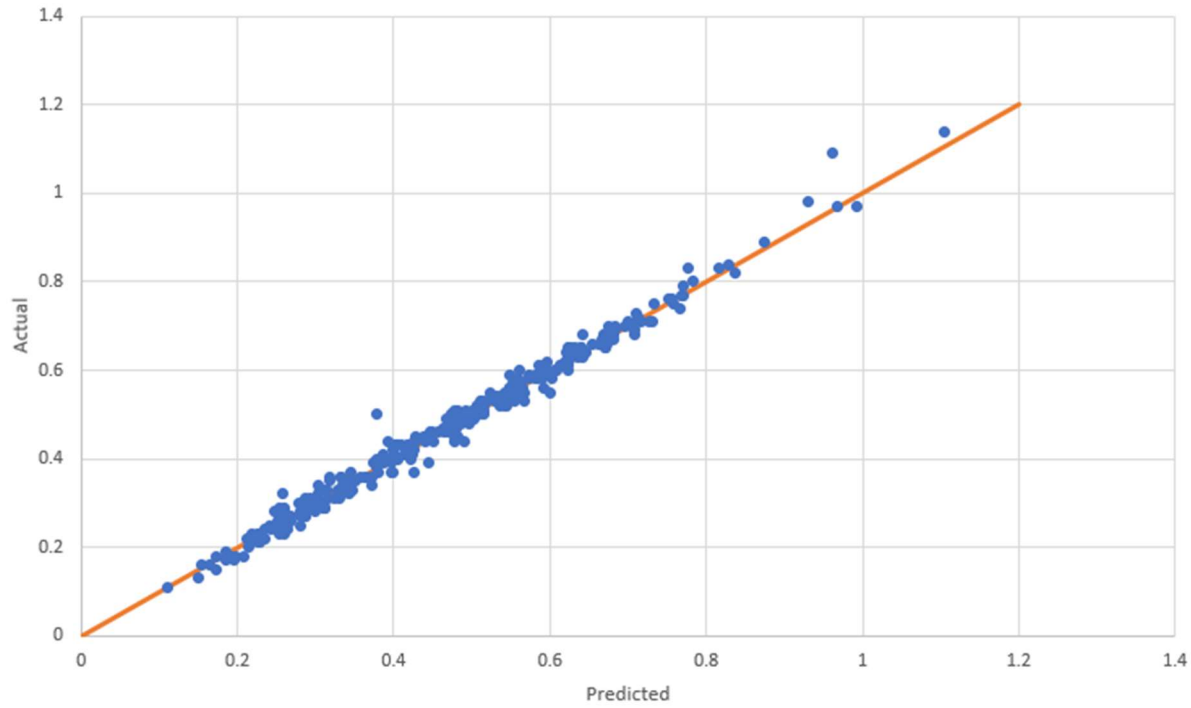
0.350699	0.35
0.751474	0.76
0.476986	0.45
0.164438	0.16
0.502695	0.49
0.412773	0.41
0.510881	0.53
0.827862	0.84
0.111516	0.11
0.482608	0.45
0.770088	0.79
0.287264	0.3
0.548528	0.56
0.265269	0.26
0.420828	0.4
0.484695	0.49
0.259358	0.23
0.673182	0.66
0.613704	0.61
0.400055	0.37
0.471347	0.46
0.531869	0.53
0.638482	0.64
0.379057	0.5
0.573251	0.58
0.253866	0.27
0.494032	0.49
0.619138	0.62
0.366619	0.36
0.703808	0.7
0.311536	0.29
0.709839	0.71
0.961477	1.09
0.641129	0.68
0.427235	0.44
0.345123	0.37
0.592987	0.6
0.514843	0.5
0.227072	0.22
0.403162	0.43
0.930521	0.98
0.287376	0.27
0.258011	0.28
0.527839	0.54

0.532046	0.53
0.731995	0.71
0.193212	0.18
0.384437	0.4
0.208142	0.18
0.195578	0.17
0.490772	0.5
0.770861	0.77
0.304162	0.34
1.104446	1.14
0.492198	0.51
0.548909	0.59
0.625651	0.65
0.337543	0.34
0.586495	0.61
0.536888	0.54
0.404405	0.4
0.252721	0.29
0.344829	0.35
0.444533	0.39
0.42635	0.44
0.681261	0.67
0.50517	0.5
0.305446	0.3
0.383195	0.39
0.381746	0.4
0.386559	0.39
0.248406	0.28
0.756266	0.76
0.218797	0.21
0.465246	0.46
0.290477	0.29
0.580977	0.58
0.282758	0.28
0.597027	0.59
0.602066	0.58
0.465542	0.46
0.46796	0.49
0.395978	0.39
0.451776	0.44
0.409749	0.43
0.235782	0.24
0.560264	0.56
0.513233	0.53

0.324752	0.31
0.314336	0.33
0.4021	0.43
0.424488	0.43
0.185712	0.19
0.478794	0.48
0.490368	0.44
0.150361	0.13
0.623576	0.6
0.684006	0.7
0.342385	0.32
0.185796	0.17
0.551717	0.57
0.597442	0.6
0.172245	0.18
0.670019	0.65
0.47984	0.47
0.662342	0.66
0.529636	0.54
0.66768	0.67
0.3834	0.4
0.396932	0.37
0.641712	0.63
0.535597	0.52
0.393648	0.44
0.422869	0.42
0.241773	0.25
0.257905	0.32
0.559516	0.55
0.42323	0.41
0.282552	0.27
0.621216	0.62
0.292407	0.31
0.70765	0.69
0.380866	0.37
0.47808	0.46
0.64053	0.65
0.635096	0.63
0.268076	0.26
0.197582	0.18
0.331844	0.36
0.285047	0.27
0.280729	0.28
0.477627	0.51

0.486457	0.48
0.815915	0.83
0.511832	0.51
0.563625	0.57
0.717272	0.71
0.606008	0.6
0.298484	0.28
0.417902	0.43
0.399456	0.42
0.506417	0.52
0.26021	0.27
0.372003	0.34
0.227687	0.21
0.463525	0.47
0.591996	0.56
0.363827	0.36
0.448345	0.46
0.626544	0.64
0.374745	0.39
0.60814	0.6
0.393301	0.4
0.643162	0.64
0.622882	0.63
0.25426	0.24
0.283385	0.28
0.542578	0.55
0.541214	0.52
0.501244	0.49
0.400744	0.41
0.280636	0.25
0.873418	0.89
0.254009	0.23
0.264913	0.24
0.278581	0.3
0.566376	0.53
0.218153	0.23
0.153482	0.16
0.599591	0.55
0.560077	0.6
0.472368	0.47
0.437799	0.45
0.519762	0.53
0.378014	0.38
0.378056	0.4

0.415673	0.41
0.530045	0.53
0.584064	0.58
0.386789	0.39
0.330943	0.31
0.695455	0.7
0.699969	0.71
0.45709	0.46
0.289939	0.3
0.528523	0.54
0.591504	0.59
0.775928	0.83
0.260672	0.25
0.317023	0.31
0.631861	0.65
0.668445	0.67
0.329144	0.33
0.514326	0.51



Bold Driver

To calculate the minimum percentage increase in order for the learning parameter to decrease, I calculated the percentage change during the overtraining session on the Basic Model

69000	0.000320092	0.00043596
69500	0.000319869	0.000435975
70000	0.000319654	0.000436004

70500	0.000319449	0.000436044
71000	0.000319252	0.000436095
71500	0.000319063	0.000436157
72000	0.000318882	0.000436228
72500	0.000318709	0.000436308
73000	0.000318543	0.000436396
73500	0.000318384	0.00043649
74000	0.000318232	0.000436591
74500	0.000318087	0.000436697
75000	0.000317947	0.000436808
75500	0.000317813	0.000436923
76000	0.000317684	0.000437041
76500	0.000317561	0.000437162
77000	0.000317443	0.000437285
77500	0.000317329	0.00043741
78000	0.00031722	0.000437536
78500	0.000317115	0.000437664
79000	0.000317014	0.000437791
79500	0.000316916	0.000437919
80000	0.000316822	0.000438047
80500	0.000316732	0.000438174
81000	0.000316644	0.0004383
81500	0.00031656	0.000438426
82000	0.000316478	0.000438551
82500	0.0003164	0.000438674
83000	0.000316323	0.000438796
83500	0.000316249	0.000438916
84000	0.000316178	0.000439035
84500	0.000316108	0.000439152
85000	0.000316041	0.000439268
85500	0.000315975	0.000439381
86000	0.000315912	0.000439493
86500	0.00031585	0.000439603
87000	0.00031579	0.00043971
87500	0.000315731	0.000439816
88000	0.000315674	0.00043992
88500	0.000315619	0.000440022
89000	0.000315565	0.000440121
89500	0.000315512	0.000440219
90000	0.00031546	0.000440315
90500	0.00031541	0.000440408
91000	0.000315361	0.0004405
91500	0.000315313	0.00044059
92000	0.000315266	0.000440677

92500	0.00031522	0.000440763
93000	0.000315175	0.000440847
93500	0.000315131	0.000440928
94000	0.000315088	0.000441008
94500	0.000315046	0.000441086
95000	0.000315005	0.000441162
95500	0.000314965	0.000441236
96000	0.000314925	0.000441309
96500	0.000314886	0.000441379
97000	0.000314848	0.000441448
97500	0.00031481	0.000441515
98000	0.000314774	0.00044158
98500	0.000314737	0.000441643
99000	0.000314702	0.000441705
99500	0.000314667	0.000441765
100000	0.000314633	0.000441824

- PercentageChange = (((0.000441824-0.00043596)/ 0.00043596)*100)/62 = 0.02169479887% increase
- 0. 000441824is the validation MSE at 100000 epochs
- 0. 00043596is the validation MSE at 33000 epochs
- 62 is derived from the fact that there is 62 rows in between those values

I will therefore using 0.02 as the increase parameter

```
@Override
public void validation(){
    super.validation();
    if(prevMse != null){
        double percentChange = ((mse-prevMse)/prevMse)*100;
        if(percentChange>=0.02){ //error function increase
            revert(); //reverts the weight and bias changes
            p.decrease();
        }else if(percentChange<=0){ //error function decrease
            p.increase();
        }
    }
    prevMse = mse;
}
```

```
BoldDriver bd = new BoldDriver(4,8,100000,new LearningParameter(0.1,0.01,0.5,0.05,0.3));
bd.run();
```

No of Epochs	Training MSE	Testing MSE
500	0.000644654	0.000873211
1000	0.000616487	0.000852148

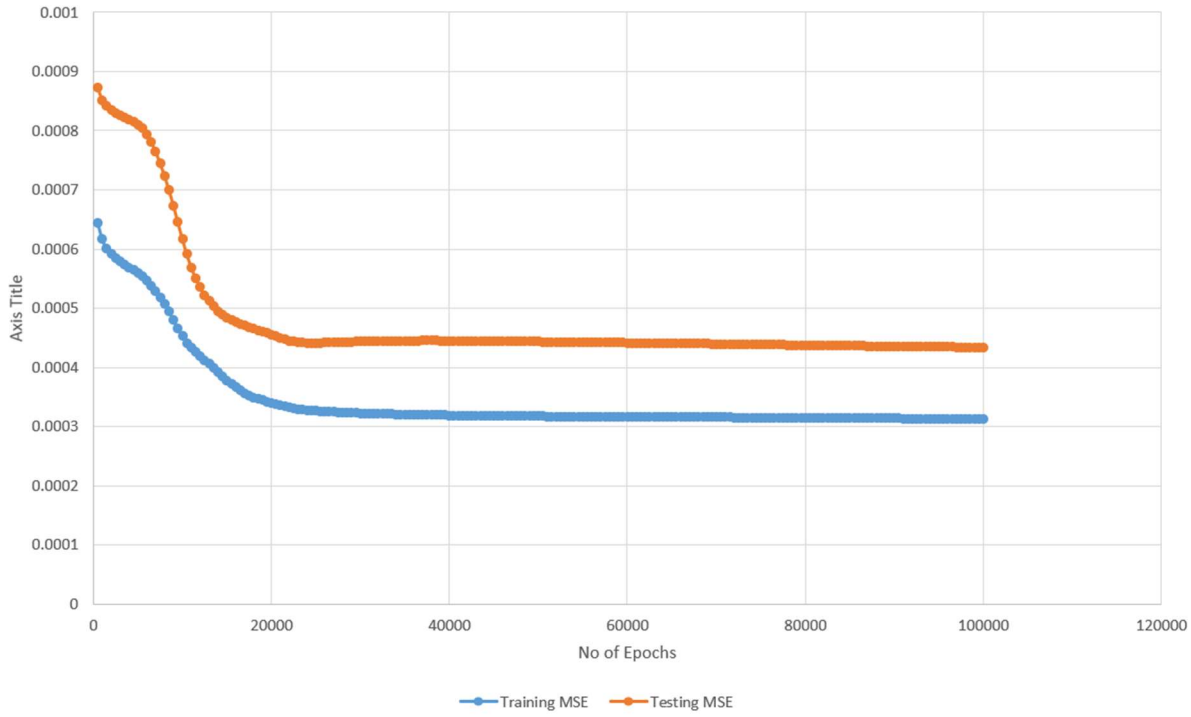
1500	0.000601367	0.000842026
2000	0.000592766	0.000835629
2500	0.000585436	0.000830233
3000	0.000579246	0.000825777
3500	0.000574014	0.000822088
4000	0.000569435	0.0008188
4500	0.000565065	0.000815309
5000	0.000560331	0.000810731
5500	0.00055461	0.000803972
6000	0.000547486	0.000794089
6500	0.000538987	0.000780813
7000	0.000529372	0.000764497
7500	0.000518754	0.000745533
8000	0.000507059	0.000724071
8500	0.000494178	0.000700079
9000	0.000480241	0.000673629
9500	0.000465981	0.000645451
10000	0.000452761	0.00061734
10500	0.000441755	0.000591531
11000	0.000433002	0.000569353
11500	0.000425671	0.000550863
12000	0.000418947	0.000535524
12500	0.000412398	0.000522772
13000	0.000405848	0.000512133
13500	0.000399179	0.000503189
14000	0.000392285	0.000495602
14500	0.000385243	0.00048921
15000	0.000378401	0.000484005
15500	0.000372131	0.000479911
16000	0.000366559	0.000476636
16500	0.000361582	0.000473732
17000	0.000357081	0.000470849
17500	0.000353111	0.000468051
18000	0.000349781	0.000465525
18500	0.00034704	0.000463183
19000	0.000344713	0.000460826
19500	0.000342631	0.000458319
20000	0.000340689	0.000455635
20500	0.000338855	0.00045285
21000	0.000337165	0.000450123
21500	0.00033499	0.000447486
22000	0.000332887	0.000445279
22500	0.000331177	0.000443636
23000	0.000329813	0.000442508

23500	0.000328734	0.000441816
24000	0.000327876	0.000441467
24500	0.000327177	0.000441372
25000	0.000326589	0.000441451
25500	0.000326078	0.00044164
26000	0.00032562	0.000441893
26500	0.000325201	0.000442174
27000	0.000324812	0.000442462
27500	0.000324447	0.000442742
28000	0.000324103	0.000443007
28500	0.000323777	0.000443255
29000	0.000323468	0.000443485
29500	0.000323173	0.000443698
30000	0.000322894	0.000443897
30500	0.000322627	0.000444082
31000	0.000322374	0.000444255
31500	0.000322133	0.000444416
32000	0.000321903	0.000444567
32500	0.000321685	0.000444707
33000	0.000321477	0.000444835
33500	0.000321278	0.000444952
34000	0.000321089	0.000445057
34500	0.000320909	0.000445149
35000	0.000320736	0.000445229
35500	0.00032057	0.000445295
36000	0.000320411	0.000445348
36500	0.000320259	0.000445387
37000	0.000320113	0.000445414
37500	0.000319972	0.000445428
38000	0.000319836	0.00044543
38500	0.000319705	0.000445421
39000	0.000319579	0.0004454
39500	0.000319457	0.000445369
40000	0.00031934	0.000445329
40500	0.000319227	0.00044528
41000	0.000319117	0.000445223
41500	0.000319012	0.000445159
42000	0.00031891	0.000445089
42500	0.000318811	0.000445013
43000	0.000318716	0.000444932
43500	0.000318624	0.000444847
44000	0.000318536	0.000444759
44500	0.00031845	0.000444668
45000	0.000318368	0.000444575

45500	0.000318288	0.00044448
46000	0.00031821	0.000444384
46500	0.000318136	0.000444287
47000	0.000318063	0.00044419
47500	0.000317993	0.000444093
48000	0.000317925	0.000443996
48500	0.000317859	0.000443899
49000	0.000317795	0.000443803
49500	0.000317733	0.000443708
50000	0.000317673	0.000443613
50500	0.000317614	0.000443519
51000	0.000317557	0.000443426
51500	0.000317501	0.000443333
52000	0.000317446	0.000443241
52500	0.000317393	0.000443149
53000	0.00031734	0.000443058
53500	0.000317289	0.000442967
54000	0.000317239	0.000442877
54500	0.00031719	0.000442786
55000	0.000317142	0.000442696
55500	0.000317094	0.000442606
56000	0.000317048	0.000442515
56500	0.000317002	0.000442425
57000	0.000316957	0.000442334
57500	0.000316912	0.000442243
58000	0.000316868	0.000442151
58500	0.000316825	0.000442059
59000	0.000316782	0.000441967
59500	0.00031674	0.000441874
60000	0.000316698	0.000441781
60500	0.000316657	0.000441687
61000	0.000316616	0.000441592
61500	0.000316575	0.000441497
62000	0.000316535	0.000441401
62500	0.000316495	0.000441305
63000	0.000316456	0.000441208
63500	0.000316417	0.000441111
64000	0.000316378	0.000441013
64500	0.000316339	0.000440914
65000	0.0003163	0.000440815
65500	0.000316262	0.000440716
66000	0.000316223	0.000440616
66500	0.000316185	0.000440516
67000	0.000316147	0.000440415

67500	0.000316109	0.000440315
68000	0.000316071	0.000440213
68500	0.000316033	0.000440112
69000	0.000315995	0.00044001
69500	0.000315957	0.000439908
70000	0.000315919	0.000439806
70500	0.000315881	0.000439704
71000	0.000315843	0.000439602
71500	0.000315805	0.000439499
72000	0.000315766	0.000439397
72500	0.000315728	0.000439294
73000	0.000315689	0.000439192
73500	0.00031565	0.000439089
74000	0.000315611	0.000438987
74500	0.000315571	0.000438885
75000	0.000315531	0.000438782
75500	0.000315491	0.00043868
76000	0.000315451	0.000438578
76500	0.00031541	0.000438476
77000	0.000315369	0.000438374
77500	0.000315327	0.000438272
78000	0.000315285	0.000438171
78500	0.000315242	0.00043807
79000	0.000315199	0.000437969
79500	0.000315155	0.000437868
80000	0.00031511	0.000437767
80500	0.000315065	0.000437667
81000	0.000315019	0.000437567
81500	0.000314973	0.000437467
82000	0.000314926	0.000437367
82500	0.000314878	0.000437268
83000	0.000314829	0.000437169
83500	0.000314779	0.00043707
84000	0.000314728	0.000436972
84500	0.000314677	0.000436874
85000	0.000314624	0.000436777
85500	0.000314571	0.00043668
86000	0.000314516	0.000436583
86500	0.00031446	0.000436487
87000	0.000314403	0.000436391
87500	0.000314345	0.000436295
88000	0.000314286	0.0004362
88500	0.000314225	0.000436106
89000	0.000314164	0.000436012

89500	0.0003141	0.000435919
90000	0.000314036	0.000435826
90500	0.00031397	0.000435733
91000	0.000313902	0.000435641
91500	0.000313834	0.00043555
92000	0.000313763	0.000435459
92500	0.000313692	0.000435368
93000	0.000313618	0.000435278
93500	0.000313543	0.000435189
94000	0.000313467	0.000435099
94500	0.000313389	0.000435011
95000	0.00031331	0.000434922
95500	0.000313229	0.000434834
96000	0.000313146	0.000434746
96500	0.000313062	0.000434658
97000	0.000312977	0.00043457
97500	0.00031289	0.000434482
98000	0.000312801	0.000434393
98500	0.000312712	0.000434305
99000	0.00031262	0.000434216
99500	0.000312528	0.000434127
100000	0.000312434	0.000434037



Overtraining occurs at around 25,000 epochs

This means that Bold Driver reaches its minima MSE for unseen data much more quickly than the basic model

It also seems to stabilize more after overtraining has reached

I believe the reason behind these two points are that initially, the MSE is rapidly decreasing, causing the learning parameter to increase from the Bold Driver, which causes slightly bigger oscillations and the potential to find the global minima much more quickly. Once the minima is found, and the algorithm continues training on the training data, it becomes too fixed to those exact values, causing the MSE for the testing data to slightly increase, which triggers the learning parameter to decrease, which would slightly stabilize the training.

So training it against 24,500 epochs:

Testing MSE: 4.4137172538335595E-4

Weights:

```
[[-0.6786803523166592, -3.0602823130281127, -0.6192460753927148, -0.2832271025951126, -3.1723690195077685, -0.6655108523961183, 0.77044068054688?1, -1.1331900712704248]
[3.177662610163048, -1.1480692305397113, -0.5125791187767795, 0.41741434779861697, 0.6706238687760938, -0.24335630343083997, 0.8852509261802639, 0.15886458166241381
[1.0069590803143125, -3.0494053753531142, -0.20212154507079932, -2.965282559242969, -1.9032674203463824, -0.7433764711099373, -0.1328567781615482, 1.3557828092283242]
[3.3875346936054656, 4.678074147189112, -0.21783462990014557, -2.255951722536571, 1.3391602297055254, 3.0683646572620633, -0.17183395434617?3, -2.0871994826?8472]
[2.4891836465709796, -5.098915063126594, 0.032888184891971976, 3.145787326454839, -2.0726056766427203, -1.877480626948228, 1.0169398796533153, 2.207765746578141]]
```

Biases:

```
[-2.266667748450851, 4.997829339281197, -0.60242862415147?5, 0.4706261419062?533, 1.1635792878340145, -2.1910000621046946, -0.0304100432434653?5, 1.8950545767650495, 1.5049277740054856]
```

Predicted Output	Actual Output
0.283954	0.27
0.631367	0.63
0.288914	0.3
0.668881	0.65
0.558432	0.55
0.548096	0.52
0.303626	0.3
0.585792	0.59
0.264528	0.29
0.643297	0.65
0.214468	0.2
0.789399	0.8
0.656585	0.66
0.738801	0.75
0.629436	0.65
0.52467	0.55
0.714781	0.73
0.625691	0.64
0.342816	0.33
0.501177	0.51
0.291579	0.31
0.390143	0.41
0.684377	0.69
0.617448	0.61
0.432639	0.45
0.171771	0.15

0.406299	0.43
0.558103	0.53
0.335757	0.32
0.59611	0.61
0.337062	0.36
0.264216	0.28
0.64925	0.64
0.49805	0.48
0.25386	0.26
0.544069	0.54
0.428816	0.42
0.51795	0.52
0.5762	0.59
0.549457	0.53
0.229659	0.23
0.271316	0.27
0.569144	0.55
0.715387	0.72
0.346044	0.35
0.568059	0.56
0.839939	0.82
0.214877	0.22
0.263521	0.24
0.769403	0.74
0.313096	0.29
0.513513	0.52
0.627191	0.62
0.674582	0.68
0.320074	0.36
0.561457	0.58
0.444477	0.46
0.313672	0.33
0.269563	0.26
0.47935	0.44
0.551558	0.56
0.289697	0.27
0.775352	0.77
0.427954	0.43
0.505343	0.52
0.484534	0.51
0.597842	0.62
0.357584	0.36
0.313951	0.29
0.761102	0.75

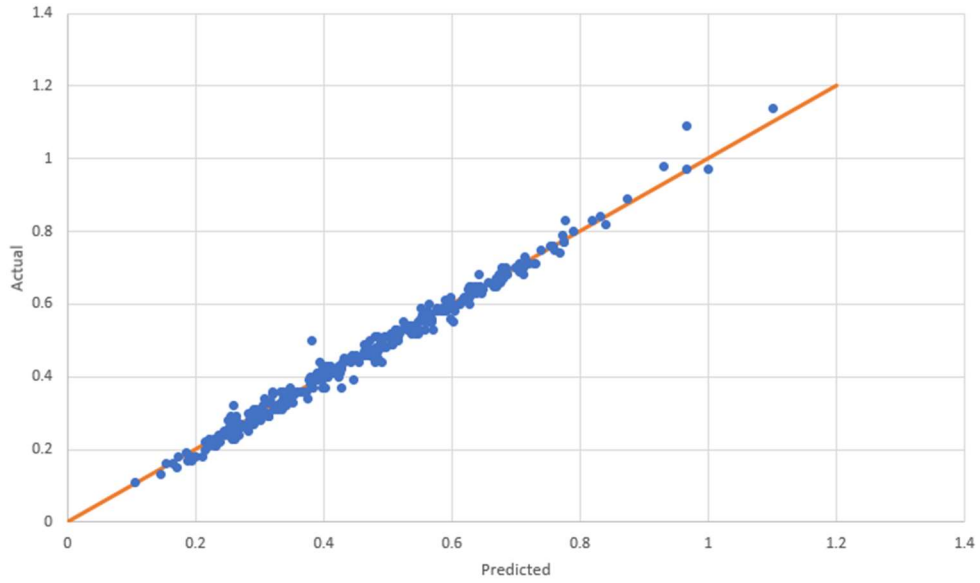
0.678299	0.7
0.592948	0.58
0.471097	0.5
0.687149	0.68
0.320852	0.35
0.351871	0.33
0.511179	0.5
0.306387	0.32
0.337318	0.34
0.332782	0.32
0.428707	0.37
0.470697	0.48
0.231444	0.21
0.373447	0.36
0.564941	0.56
0.246074	0.24
0.72781	0.71
0.229668	0.23
0.626012	0.61
0.967239	0.97
0.311062	0.3
0.711947	0.68
0.238622	0.22
0.442308	0.44
0.999708	0.97
0.600077	0.6
0.353988	0.35
0.754768	0.76
0.479136	0.45
0.165719	0.16
0.507304	0.49
0.406463	0.41
0.512604	0.53
0.831658	0.84
0.105746	0.11
0.484981	0.45
0.773665	0.79
0.290528	0.3
0.551623	0.56
0.268517	0.26
0.422547	0.4
0.488507	0.49
0.261772	0.23
0.676468	0.66

0.616934	0.61
0.402915	0.37
0.473683	0.46
0.530008	0.53
0.640722	0.64
0.38057	0.5
0.577282	0.58
0.256714	0.27
0.493851	0.49
0.622961	0.62
0.371017	0.36
0.711715	0.7
0.314434	0.29
0.71248	0.71
0.966145	1.09
0.643044	0.68
0.432684	0.44
0.34819	0.37
0.595205	0.6
0.516903	0.5
0.229642	0.22
0.407415	0.43
0.930589	0.98
0.291757	0.27
0.261552	0.28
0.529142	0.54
0.536527	0.53
0.73159	0.71
0.194841	0.18
0.3895	0.4
0.210525	0.18
0.195177	0.17
0.489523	0.5
0.774872	0.77
0.307511	0.34
1.101897	1.14
0.496172	0.51
0.551531	0.59
0.628598	0.65
0.340109	0.34
0.590237	0.61
0.538199	0.54
0.406223	0.4
0.255628	0.29

0.347212	0.35
0.4459	0.39
0.429259	0.44
0.681018	0.67
0.507345	0.5
0.307087	0.3
0.385194	0.39
0.386133	0.4
0.388486	0.39
0.251113	0.28
0.757946	0.76
0.221771	0.21
0.470038	0.46
0.292738	0.29
0.582926	0.58
0.285263	0.28
0.600837	0.59
0.604229	0.58
0.464595	0.46
0.464275	0.49
0.397896	0.39
0.45572	0.44
0.411653	0.43
0.235583	0.24
0.564688	0.56
0.515071	0.53
0.32838	0.31
0.31784	0.33
0.404044	0.43
0.426383	0.43
0.185796	0.19
0.48216	0.48
0.491012	0.44
0.145626	0.13
0.627505	0.6
0.685319	0.7
0.341937	0.32
0.188102	0.17
0.555061	0.57
0.601304	0.6
0.172708	0.18
0.665959	0.65
0.484459	0.47
0.666306	0.66

0.532459	0.54
0.670182	0.67
0.386072	0.4
0.398016	0.37
0.646803	0.63
0.536744	0.52
0.394565	0.44
0.424991	0.42
0.245347	0.25
0.259906	0.32
0.564367	0.55
0.425099	0.41
0.285585	0.27
0.618791	0.62
0.294787	0.31
0.706532	0.69
0.38265	0.37
0.482353	0.46
0.644636	0.65
0.638182	0.63
0.271768	0.26
0.200198	0.18
0.333494	0.36
0.287296	0.27
0.284758	0.28
0.4802	0.51
0.491433	0.48
0.818508	0.83
0.511189	0.51
0.566517	0.57
0.719381	0.71
0.610301	0.6
0.301166	0.28
0.422856	0.43
0.397863	0.42
0.505353	0.52
0.26334	0.27
0.37482	0.34
0.230111	0.21
0.463919	0.47
0.59752	0.56
0.364909	0.36
0.451216	0.46
0.630293	0.64

0.378296	0.39
0.612292	0.6
0.397939	0.4
0.644663	0.64
0.626758	0.63
0.256945	0.24
0.286238	0.28
0.546987	0.55
0.544205	0.52
0.505858	0.49
0.405017	0.41
0.283393	0.25
0.87305	0.89
0.256834	0.23
0.267174	0.24
0.281787	0.3
0.56995	0.53
0.220854	0.23
0.154454	0.16
0.603127	0.55
0.56379	0.6
0.477073	0.47
0.44069	0.45
0.522747	0.53
0.379777	0.38
0.379246	0.4
0.414963	0.41
0.533986	0.53
0.587274	0.58
0.391384	0.39
0.336207	0.31
0.698275	0.7
0.704688	0.71
0.460558	0.46
0.293261	0.3
0.531379	0.54
0.596016	0.59
0.776283	0.83
0.263315	0.25
0.322136	0.31
0.635517	0.65
0.670808	0.67
0.332173	0.33
0.516314	0.51



Annealing

```
public double p(int x){
    return ep+(p-ep)*(1-(1/(1+Math.exp(10-((20*(x))/noEpochs))))); //returns learning parameter based on current number of epochs
}
```

```
Annealing a = new Annealing(4,8,100000,0.1,0.01);
a.run();
```

No of Epochs	Training MSE	Testing MSE
500	0.000707584	0.000927887
1000	0.000655024	0.000885486
1500	0.000627597	0.000864693
2000	0.000612679	0.000854158
2500	0.000603641	0.000847823
3000	0.000596977	0.000842659
3500	0.000591085	0.000837439
4000	0.000585269	0.000831706
4500	0.00057916	0.000825179
5000	0.000572475	0.000817505
5500	0.000565044	0.00080833
6000	0.000557059	0.000797724
6500	0.000549175	0.000786503
7000	0.000542104	0.000775781
7500	0.00053614	0.00076623
8000	0.000531168	0.000757919
8500	0.000526932	0.000750617
9000	0.000523197	0.000744051
9500	0.000519795	0.000737997

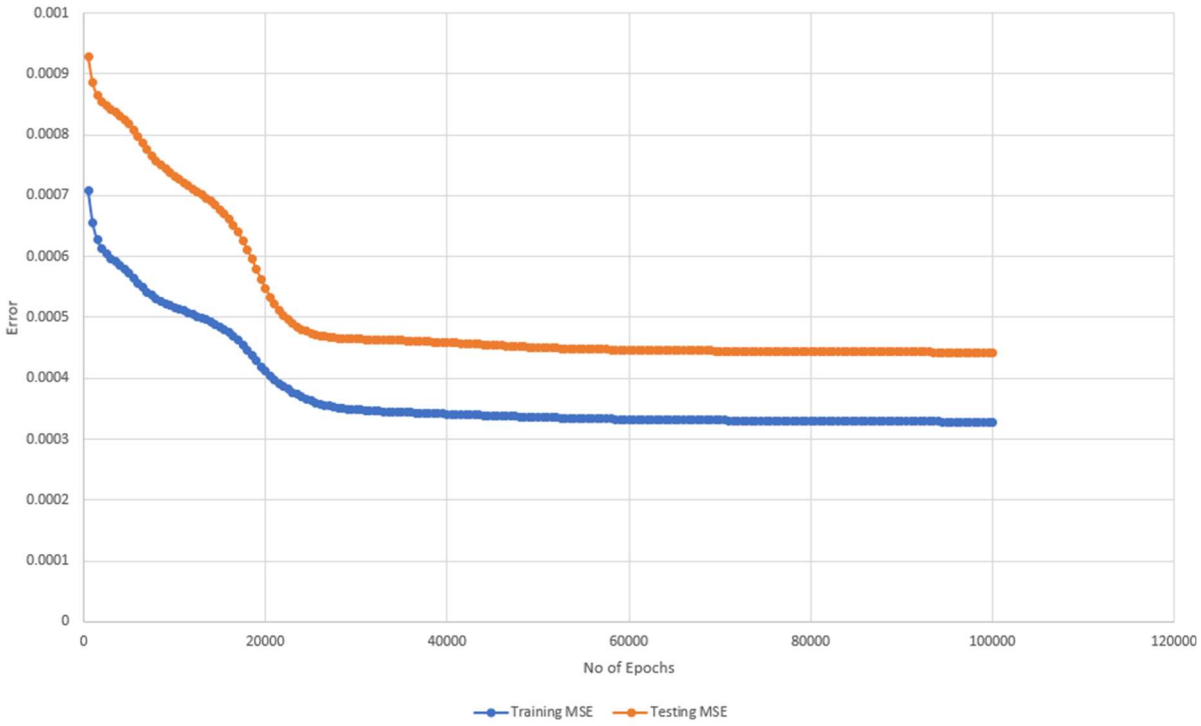
10000	0.000516616	0.000732307
10500	0.000513585	0.000726882
11000	0.000510653	0.000721649
11500	0.000507776	0.000716549
12000	0.000504913	0.000711519
12500	0.000502016	0.000706485
13000	0.000499033	0.000701361
13500	0.000495898	0.000696042
14000	0.000492537	0.0006904
14500	0.000488861	0.000684286
15000	0.000484771	0.000677522
15500	0.000480155	0.000669904
16000	0.0004749	0.000661201
16500	0.000468892	0.000651175
17000	0.000462052	0.000639616
17500	0.00045437	0.000626412
18000	0.000445955	0.000611634
18500	0.000437063	0.000595635
19000	0.000428084	0.00057906
19500	0.00041944	0.00056274
20000	0.000411449	0.000547462
20500	0.000404251	0.00053378
21000	0.000397802	0.000521883
21500	0.000391976	0.000511727
22000	0.000386652	0.000503136
22500	0.000381746	0.000495899
23000	0.000377219	0.000489818
23500	0.000373062	0.000484729
24000	0.000369281	0.000480495
24500	0.000365881	0.000477001
25000	0.000362863	0.000474146
25500	0.000360224	0.000471848
26000	0.000357935	0.00047001
26500	0.000355964	0.000468556
27000	0.000354279	0.000467417
27500	0.000352843	0.000466529
28000	0.000351619	0.00046584
28500	0.000350574	0.000465302
29000	0.000349677	0.000464878
29500	0.000348901	0.000464538
30000	0.000348224	0.000464255
30500	0.000347632	0.000464009
31000	0.000347104	0.000463791
31500	0.000346627	0.000463585

32000	0.00034619	0.000463383
32500	0.000345786	0.000463177
33000	0.000345409	0.000462965
33500	0.000345052	0.000462742
34000	0.000344713	0.000462505
34500	0.000344387	0.000462254
35000	0.000344073	0.000461988
35500	0.000343774	0.000461698
36000	0.000343482	0.000461409
36500	0.000343195	0.000461105
37000	0.000342912	0.000460785
37500	0.000342631	0.000460451
38000	0.000342351	0.000460103
38500	0.000342072	0.000459741
39000	0.000341794	0.000459366
39500	0.000341514	0.000458979
40000	0.000341234	0.00045858
40500	0.000340971	0.000458166
41000	0.000340706	0.000457777
41500	0.000340439	0.000457379
42000	0.000340169	0.000456973
42500	0.000339897	0.000456559
43000	0.000339621	0.000456137
43500	0.000339342	0.000455708
44000	0.00033906	0.000455273
44500	0.000338774	0.000454831
45000	0.000338483	0.000454383
45500	0.000338228	0.000453968
46000	0.000337975	0.00045359
46500	0.000337719	0.000453205
47000	0.000337459	0.000452817
47500	0.000337196	0.000452425
48000	0.000336929	0.00045203
48500	0.000336659	0.000451633
49000	0.000336386	0.000451233
49500	0.000336109	0.000450831
50000	0.000335828	0.000450429
50500	0.000335604	0.000450223
51000	0.000335396	0.000449949
51500	0.000335186	0.000449669
52000	0.000334974	0.000449387
52500	0.000334761	0.000449103
53000	0.000334546	0.000448818
53500	0.00033433	0.000448533

54000	0.000334111	0.000448247
54500	0.000333891	0.000447962
55000	0.00033367	0.000447676
55500	0.000333512	0.000447742
56000	0.000333373	0.000447591
56500	0.000333233	0.000447429
57000	0.000333092	0.000447264
57500	0.000332952	0.000447099
58000	0.00033281	0.000446933
58500	0.000332669	0.000446767
59000	0.000332526	0.000446602
59500	0.000332384	0.000446436
60000	0.000332241	0.00044627
60500	0.000332148	0.000446302
61000	0.000332061	0.000446222
61500	0.000331974	0.000446131
62000	0.000331886	0.000446038
62500	0.000331799	0.000445943
63000	0.000331711	0.000445849
63500	0.000331624	0.000445754
64000	0.000331536	0.000445659
64500	0.000331448	0.000445564
65000	0.00033136	0.00044547
65500	0.000331298	0.000445429
66000	0.000331237	0.000445375
66500	0.000331177	0.000445315
67000	0.000331116	0.000445253
67500	0.000331055	0.00044519
68000	0.000330994	0.000445127
68500	0.000330933	0.000445064
69000	0.000330872	0.000445
69500	0.000330812	0.000444937
70000	0.000330751	0.000444874
70500	0.000330701	0.000444827
71000	0.000330651	0.000444781
71500	0.000330601	0.000444732
72000	0.000330551	0.000444682
72500	0.000330502	0.000444632
73000	0.000330452	0.000444581
73500	0.000330402	0.000444531
74000	0.000330352	0.00044448
74500	0.000330303	0.00044443
75000	0.000330253	0.000444379
75500	0.000330207	0.000444335

76000	0.000330162	0.000444291
76500	0.000330116	0.000444246
77000	0.000330071	0.000444201
77500	0.000330025	0.000444156
78000	0.00032998	0.00044411
78500	0.000329934	0.000444065
79000	0.000329889	0.00044402
79500	0.000329843	0.000443975
80000	0.000329798	0.00044393
80500	0.000329754	0.000443887
81000	0.00032971	0.000443844
81500	0.000329666	0.000443802
82000	0.000329622	0.000443759
82500	0.000329579	0.000443716
83000	0.000329535	0.000443673
83500	0.000329491	0.00044363
84000	0.000329447	0.000443588
84500	0.000329403	0.000443545
85000	0.000329359	0.000443503
85500	0.000329316	0.000443461
86000	0.000329273	0.000443419
86500	0.00032923	0.000443378
87000	0.000329186	0.000443337
87500	0.000329143	0.000443295
88000	0.0003291	0.000443254
88500	0.000329057	0.000443213
89000	0.000329013	0.000443172
89500	0.00032897	0.000443131
90000	0.000328927	0.00044309
90500	0.000328884	0.00044305
91000	0.000328841	0.000443009
91500	0.000328798	0.000442969
92000	0.000328755	0.000442929
92500	0.000328713	0.000442889
93000	0.00032867	0.000442849
93500	0.000328627	0.00044281
94000	0.000328584	0.00044277
94500	0.000328541	0.00044273
95000	0.000328498	0.000442691
95500	0.000328456	0.000442652
96000	0.000328413	0.000442613
96500	0.00032837	0.000442574
97000	0.000328328	0.000442535
97500	0.000328285	0.000442497

98000	0.000328242	0.000442458
98500	0.0003282	0.00044242
99000	0.000328157	0.000442381
99500	0.000328115	0.000442343
100000	0.000328072	0.000442305



After 100,000 epochs, it reaches a MSE of 0.000442305

Comparing it with the Basic Model

Testing MSE: 4.4230541128348116E-4

```
Weights:
[[-0.013033258910637504, 2.864665080149935, -0.5330646532851356, -2.5429811062589334, -0.4823535036236117, 1.2610052541502 71, 0.41530807575462125, 0.09888184032534234]
[0.7990788633736189, 1.1758001947537036, -0.10365816498239558, 0.4175190555234655, 0.4068421848193785, -0.6138385552641759, -3.126555794488805, -0.12146418999433296]
[-2.786674898142886, 3.0332143506758955, -1.1125200045458 92, -1.9118306595666787, -0.5595174912893872, -1.3388553664821442, -0.5433126604396403, 0.2186758962155601 6]
[-2.002439731029969, -4.43251189904255, 0.5711187076653871, 0.42919027980301355, 3.3538796126686594, 1.4055168995865164, -2.237061255831218, -0.5056873692655726]
[3.4168855151531647, 5.234545994605135, -0.96799057963327, -2.2304302129980065, -2.651423808134911, -1.5694705454516837, -2.553100834642129, 0.9478278583736611]]

Biases:
[0.006424032886854873, -5.182015975899 33, 0.483133788181829, 1.668077133323282, -3.0631062093342933, -1.4606709085784688, 1.5997496542718352, -0.04921356941419742, 1.850282732572557 5]
```

Predicted Output	Actual Output
0.281247	0.27
0.632738	0.63
0.287771	0.3
0.667729	0.65
0.555995	0.55
0.547068	0.52
0.304416	0.3
0.582566	0.59
0.262399	0.29

0.642044	0.65
0.21339	0.2
0.785047	0.8
0.654991	0.66
0.733121	0.75
0.627923	0.65
0.520481	0.55
0.713414	0.73
0.620339	0.64
0.343131	0.33
0.492001	0.51
0.293475	0.31
0.386296	0.41
0.682904	0.69
0.620876	0.61
0.428664	0.45
0.169928	0.15
0.403473	0.43
0.555025	0.53
0.332619	0.32
0.592994	0.61
0.335789	0.36
0.262318	0.28
0.64837	0.64
0.492444	0.48
0.25216	0.26
0.539215	0.54
0.426826	0.42
0.515118	0.52
0.574847	0.59
0.544763	0.53
0.227673	0.23
0.27008	0.27
0.566363	0.55
0.712718	0.72
0.343609	0.35
0.567199	0.56
0.837377	0.82
0.213419	0.22
0.263239	0.24
0.764989	0.74
0.311546	0.29
0.511751	0.52
0.629285	0.62

0.670727	0.68
0.318344	0.36
0.556667	0.58
0.444259	0.46
0.315038	0.33
0.266754	0.26
0.475757	0.44
0.54742	0.56
0.28762	0.27
0.769993	0.77
0.427509	0.43
0.501874	0.52
0.487552	0.51
0.594251	0.62
0.358893	0.36
0.311129	0.29
0.760002	0.75
0.675784	0.7
0.5889	0.58
0.474635	0.5
0.686975	0.68
0.319834	0.35
0.348963	0.33
0.508555	0.5
0.305015	0.32
0.334672	0.34
0.330359	0.32
0.42548	0.37
0.4732	0.48
0.229537	0.21
0.371663	0.36
0.563623	0.56
0.244835	0.24
0.726206	0.71
0.229063	0.23
0.623731	0.61
0.960021	0.97
0.308616	0.3
0.709754	0.68
0.236827	0.22
0.437348	0.44
0.998316	0.97
0.59706	0.6
0.352528	0.35

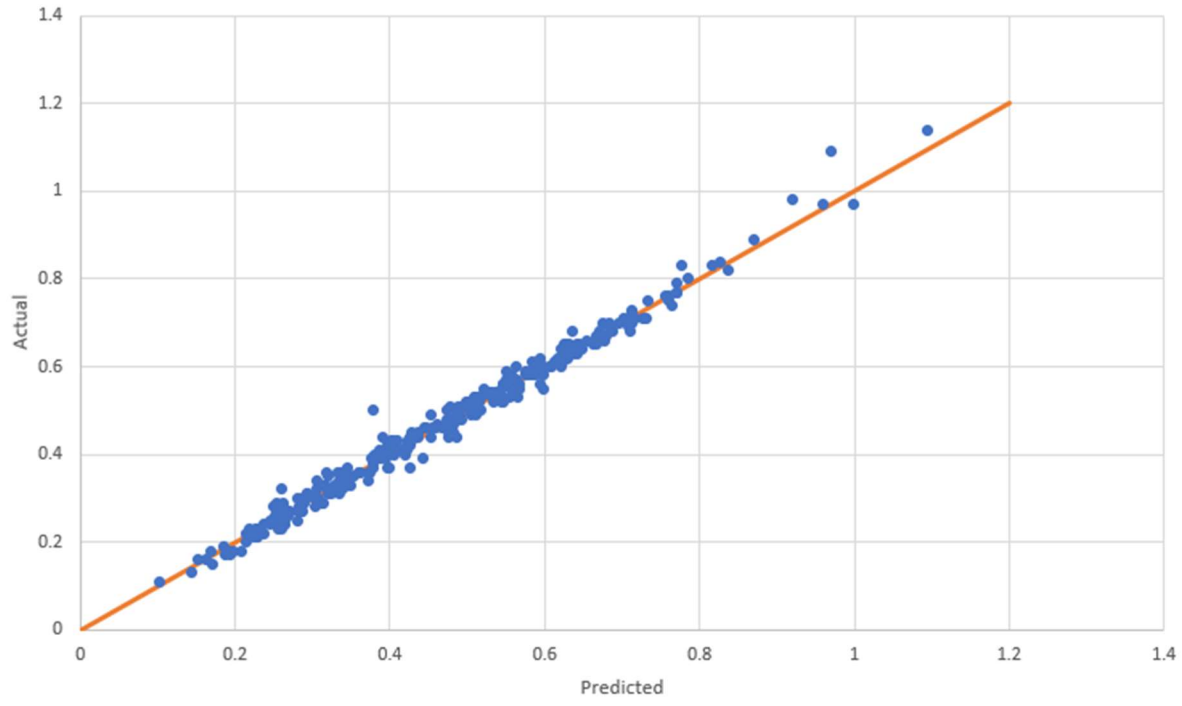
0.75636	0.76
0.475508	0.45
0.162534	0.16
0.505043	0.49
0.410283	0.41
0.508723	0.53
0.826711	0.84
0.102154	0.11
0.481617	0.45
0.770574	0.79
0.289692	0.3
0.546008	0.56
0.266107	0.26
0.419317	0.4
0.485385	0.49
0.259107	0.23
0.676731	0.66
0.613053	0.61
0.399151	0.37
0.471275	0.46
0.518283	0.53
0.639837	0.64
0.37852	0.5
0.575396	0.58
0.254486	0.27
0.489782	0.49
0.618026	0.62
0.3743	0.36
0.71166	0.7
0.313448	0.29
0.7138	0.71
0.968984	1.09
0.635355	0.68
0.431164	0.44
0.346083	0.37
0.591263	0.6
0.516727	0.5
0.226958	0.22
0.40374	0.43
0.921044	0.98
0.285191	0.27
0.259827	0.28
0.529948	0.54
0.535604	0.53

0.73031	0.71
0.192632	0.18
0.386806	0.4
0.208397	0.18
0.193758	0.17
0.485443	0.5
0.770163	0.77
0.305575	0.34
1.094268	1.14
0.494154	0.51
0.550961	0.59
0.62571	0.65
0.338506	0.34
0.584225	0.61
0.536671	0.54
0.404473	0.4
0.254164	0.29
0.34503	0.35
0.442488	0.39
0.425095	0.44
0.679124	0.67
0.504122	0.5
0.304278	0.3
0.381901	0.39
0.387952	0.4
0.38618	0.39
0.248638	0.28
0.760461	0.76
0.220216	0.21
0.467129	0.46
0.290005	0.29
0.579205	0.58
0.283033	0.28
0.596473	0.59
0.598939	0.58
0.459865	0.46
0.453479	0.49
0.395267	0.39
0.453264	0.44
0.408748	0.43
0.236225	0.24
0.562624	0.56
0.511808	0.53
0.323965	0.31

0.315714	0.33
0.402546	0.43
0.424118	0.43
0.184613	0.19
0.47772	0.48
0.486655	0.44
0.14363	0.13
0.62158	0.6
0.682925	0.7
0.33945	0.32
0.18765	0.17
0.550793	0.57
0.59882	0.6
0.169209	0.18
0.663468	0.65
0.481393	0.47
0.667791	0.66
0.529845	0.54
0.669917	0.67
0.384703	0.4
0.396951	0.37
0.64035	0.63
0.53388	0.52
0.391068	0.44
0.422171	0.42
0.245067	0.25
0.260422	0.32
0.562794	0.55
0.422704	0.41
0.284856	0.27
0.617596	0.62
0.292235	0.31
0.707287	0.69
0.379256	0.37
0.478743	0.46
0.642995	0.65
0.641754	0.63
0.268916	0.26
0.19826	0.18
0.333267	0.36
0.284486	0.27
0.28442	0.28
0.478181	0.51
0.489986	0.48

0.816925	0.83
0.507829	0.51
0.563387	0.57
0.717432	0.71
0.606913	0.6
0.302552	0.28
0.422932	0.43
0.396825	0.42
0.499102	0.52
0.261258	0.27
0.371858	0.34
0.227452	0.21
0.461299	0.47
0.594966	0.56
0.361095	0.36
0.44677	0.46
0.625578	0.64
0.37589	0.39
0.60844	0.6
0.395279	0.4
0.644259	0.64
0.623124	0.63
0.252779	0.24
0.283917	0.28
0.544839	0.55
0.543397	0.52
0.510383	0.49
0.401272	0.41
0.281033	0.25
0.870198	0.89
0.25538	0.23
0.264855	0.24
0.281034	0.3
0.564467	0.53
0.21884	0.23
0.151336	0.16
0.597693	0.55
0.56331	0.6
0.474444	0.47
0.435676	0.45
0.517831	0.53
0.377271	0.38
0.380958	0.4
0.412336	0.41

0.531002	0.53
0.583484	0.58
0.388801	0.39
0.334531	0.31
0.695636	0.7
0.702012	0.71
0.4561	0.46
0.291769	0.3
0.525205	0.54
0.59398	0.59
0.777576	0.83
0.261788	0.25
0.320658	0.31
0.630725	0.65
0.667428	0.67
0.329039	0.33
0.512256	0.51



Weight Decay

```
WeightDecay wd = new WeightDecay(4,8,100000,0.1);
wd.run();
```

No of Epochs	Training MSE	Validation MSE
500	0.000838951	0.001023518
1000	0.000698938	0.000912571

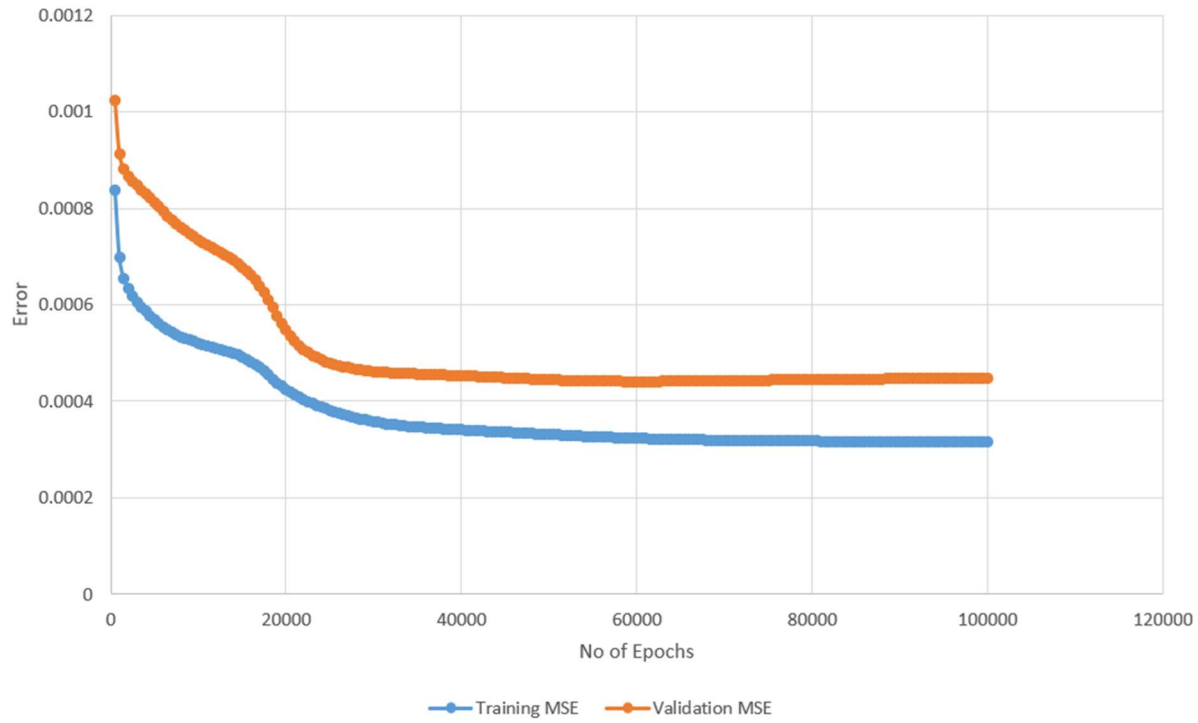
1500	0.000654841	0.000881363
2000	0.000632732	0.000866384
2500	0.000618117	0.0008559
3000	0.000606477	0.000846771
3500	0.000596223	0.000838189
4000	0.000586717	0.000829848
4500	0.000577646	0.000821394
5000	0.000568935	0.000812483
5500	0.000560805	0.000803126
6000	0.000553605	0.000793731
6500	0.000547488	0.000784727
7000	0.00054233	0.000776295
7500	0.000537896	0.000768419
8000	0.000533976	0.000761032
8500	0.000530425	0.000754073
9000	0.000527149	0.0007475
9500	0.000524084	0.000741281
10000	0.000521188	0.000735383
10500	0.000518423	0.000729766
11000	0.000515753	0.000724379
11500	0.00051314	0.000719157
12000	0.000510541	0.000714022
12500	0.000507904	0.000708884
13000	0.000505172	0.000703637
13500	0.000502276	0.000698156
14000	0.000499137	0.000692299
14500	0.000495664	0.000685899
15000	0.000491758	0.000678765
15500	0.00048731	0.000670685
16000	0.000482215	0.000661432
16500	0.000476386	0.000650794
17000	0.000469784	0.000638621
17500	0.000462464	0.000624904
18000	0.000454606	0.000609858
18500	0.000446523	0.000593968
19000	0.000438599	0.000577943
19500	0.000431183	0.000562548
20000	0.000424477	0.00054841
20500	0.000418507	0.000535879
21000	0.000413172	0.000525035
21500	0.000408326	0.000515772
22000	0.00040384	0.000507896
22500	0.000399625	0.000501197
23000	0.00039563	0.00049548

23500	0.000391831	0.000490577
24000	0.000388217	0.000486345
24500	0.000384784	0.000482667
25000	0.000381528	0.00047945
25500	0.000378445	0.000476618
26000	0.00037553	0.000474114
26500	0.00037278	0.000471891
27000	0.000370189	0.000469916
27500	0.000367756	0.000468161
28000	0.000365475	0.000466604
28500	0.000363346	0.000465226
29000	0.000361363	0.000464009
29500	0.000359524	0.000462937
30000	0.000357823	0.000461993
30500	0.000356254	0.000461163
31000	0.000354808	0.00046043
31500	0.000353479	0.00045978
32000	0.000352258	0.000459202
32500	0.000351135	0.000458682
33000	0.000350101	0.000458209
33500	0.000349148	0.000457775
34000	0.000348267	0.00045737
34500	0.00034745	0.000456986
35000	0.000346689	0.000456618
35500	0.000345976	0.00045626
36000	0.000345307	0.000455907
36500	0.000344674	0.000455556
37000	0.000344072	0.000455203
37500	0.000343497	0.000454846
38000	0.000342945	0.000454483
38500	0.000342411	0.000454112
39000	0.000341892	0.000453734
39500	0.000341385	0.000453347
40000	0.000340888	0.000452952
40500	0.000340398	0.000452548
41000	0.000339915	0.000452137
41500	0.000339435	0.000451719
42000	0.000338959	0.000451295
42500	0.000338484	0.000450865
43000	0.00033801	0.000450432
43500	0.000337536	0.000449996
44000	0.000337062	0.000449559
44500	0.000336588	0.000449121
45000	0.000336113	0.000448686

45500	0.000335636	0.000448253
46000	0.00033516	0.000447824
46500	0.000334682	0.0004474
47000	0.000334204	0.000446984
47500	0.000333726	0.000446576
48000	0.000333249	0.000446177
48500	0.000332772	0.000445789
49000	0.000332297	0.000445412
49500	0.000331824	0.000445048
50000	0.000331354	0.000444698
50500	0.000330887	0.000444363
51000	0.000330424	0.000444043
51500	0.000329965	0.000443739
52000	0.000329512	0.000443451
52500	0.000329065	0.000443181
53000	0.000328624	0.000442928
53500	0.000328191	0.000442693
54000	0.000327765	0.000442477
54500	0.000327348	0.000442278
55000	0.00032694	0.000442097
55500	0.000326541	0.000441935
56000	0.000326152	0.00044179
56500	0.000325774	0.000441663
57000	0.000325406	0.000441554
57500	0.000325048	0.000441461
58000	0.000324702	0.000441384
58500	0.000324367	0.000441324
59000	0.000324043	0.000441279
59500	0.00032373	0.000441248
60000	0.000323428	0.000441232
60500	0.000323138	0.000441228
61000	0.000322858	0.000441238
61500	0.000322589	0.000441259
62000	0.000322331	0.000441291
62500	0.000322084	0.000441333
63000	0.000321846	0.000441385
63500	0.000321618	0.000441446
64000	0.0003214	0.000441515
64500	0.000321191	0.000441591
65000	0.000320991	0.000441674
65500	0.000320799	0.000441763
66000	0.000320615	0.000441857
66500	0.000320439	0.000441956
67000	0.000320271	0.000442059

67500	0.000320109	0.000442166
68000	0.000319955	0.000442276
68500	0.000319806	0.000442388
69000	0.000319664	0.000442503
69500	0.000319527	0.000442619
70000	0.000319396	0.000442737
70500	0.00031927	0.000442857
71000	0.000319148	0.000442976
71500	0.000319031	0.000443097
72000	0.000318919	0.000443217
72500	0.00031881	0.000443338
73000	0.000318706	0.000443458
73500	0.000318605	0.000443578
74000	0.000318507	0.000443697
74500	0.000318413	0.000443816
75000	0.000318322	0.000443934
75500	0.000318233	0.00044405
76000	0.000318147	0.000444166
76500	0.000318064	0.00044428
77000	0.000317984	0.000444394
77500	0.000317905	0.000444505
78000	0.000317829	0.000444616
78500	0.000317755	0.000444725
79000	0.000317683	0.000444832
79500	0.000317613	0.000444938
80000	0.000317544	0.000445043
80500	0.000317478	0.000445146
81000	0.000317413	0.000445247
81500	0.000317349	0.000445347
82000	0.000317287	0.000445445
82500	0.000317227	0.000445541
83000	0.000317167	0.000445636
83500	0.00031711	0.000445729
84000	0.000317053	0.00044582
84500	0.000316998	0.00044591
85000	0.000316943	0.000445998
85500	0.00031689	0.000446085
86000	0.000316838	0.00044617
86500	0.000316787	0.000446253
87000	0.000316737	0.000446335
87500	0.000316688	0.000446415
88000	0.00031664	0.000446493
88500	0.000316593	0.00044657
89000	0.000316546	0.000446646

89500	0.000316501	0.000446719
90000	0.000316456	0.000446792
90500	0.000316412	0.000446862
91000	0.000316369	0.000446932
91500	0.000316326	0.000446999
92000	0.000316284	0.000447066
92500	0.000316243	0.00044713
93000	0.000316203	0.000447194
93500	0.000316163	0.000447256
94000	0.000316124	0.000447316
94500	0.000316085	0.000447375
95000	0.000316047	0.000447433
95500	0.000316009	0.00044749
96000	0.000315972	0.000447545
96500	0.000315936	0.000447598
97000	0.0003159	0.000447651
97500	0.000315865	0.000447702
98000	0.00031583	0.000447752
98500	0.000315795	0.000447801
99000	0.000315761	0.000447848
99500	0.000315728	0.000447894
100000	0.000315695	0.000447939



Overtraining starts at 64,000 epochs

I will therefore train the model for 63,500 epochs:

```
WeightDecay wd = new WeightDecay(4,8,63500,0.1);
wd.run();
```

Testing MSE: 4.41445957280988E-4

```
Weights:
[[-0.3835065235590646, 2.9203815074367174, 0.5026518637490442, -3.1282325593214306, -0.4792772407502057, 0.16359069027246920, 0.6402681368834002, 0.21065006625747762]
[0.7073978834621718, 1.2261161185600059, -0.5696428960820922, 0.5789045868779507, 0.3567300652576928, -0.2499058532137172 8, -3.0642024676052040, -0.2970847540487336 3]
[-2.840746489392122, 2.9980821083426923, -1.3690925125971565, -1.6185559942151668, -0.6369863124266363, -0.7920022213046067, -0.6655568795165416, 0.3711439436198554]
[-1.9208541321138124, -4.650943667697094, 3.0233765716608754, 1.1660843743816451, 1.5543392082707093, 0.7837031022333318, -3.2702776418958797, -0.6305328962047759]
[3.5102720788052704, 5.775169105110222, -2.5193637700207248, -2.186990701671041, -1.3623214207084475, -0.4702657035984329, -2.6723904331476533, 1.0814033577295814]]

Biases:
[0.10249260635241754, -5.092769948973918, -1.9072142270236443, 1.1176523162431173, -1.590814578365140, -0.8447961056008094, 2.0551971459449163, -0.1548367183234668, 1.5590257536808795]
```

Predicted Output	Actual Output
0.281987	0.27
0.631319	0.63
0.287562	0.3
0.666765	0.65
0.556095	0.55
0.547197	0.52
0.302027	0.3
0.58334	0.59
0.262099	0.29
0.641392	0.65
0.217502	0.2
0.788007	0.8
0.655282	0.66
0.736118	0.75
0.626511	0.65
0.521968	0.55
0.713139	0.73
0.622886	0.64
0.342559	0.33
0.497841	0.51
0.290108	0.31
0.387552	0.41
0.683942	0.69
0.6171	0.61
0.429725	0.45
0.174249	0.15
0.403773	0.43
0.555517	0.53
0.33371	0.32
0.594046	0.61
0.33539	0.36
0.263061	0.28
0.64869	0.64
0.49551	0.48

0.252348	0.26
0.541681	0.54
0.427672	0.42
0.515733	0.52
0.575432	0.59
0.546775	0.53
0.228003	0.23
0.270232	0.27
0.567352	0.55
0.714424	0.72
0.344177	0.35
0.567777	0.56
0.839922	0.82
0.213124	0.22
0.26255	0.24
0.767543	0.74
0.312199	0.29
0.511538	0.52
0.627853	0.62
0.672525	0.68
0.317797	0.36
0.558602	0.58
0.443888	0.46
0.311454	0.33
0.267706	0.26
0.479249	0.44
0.548668	0.56
0.287983	0.27
0.773253	0.77
0.426141	0.43
0.50338	0.52
0.485509	0.51
0.596556	0.62
0.357737	0.36
0.311623	0.29
0.761458	0.75
0.676893	0.7
0.590859	0.58
0.471557	0.5
0.686372	0.68
0.318729	0.35
0.349897	0.33
0.510468	0.5
0.304623	0.32

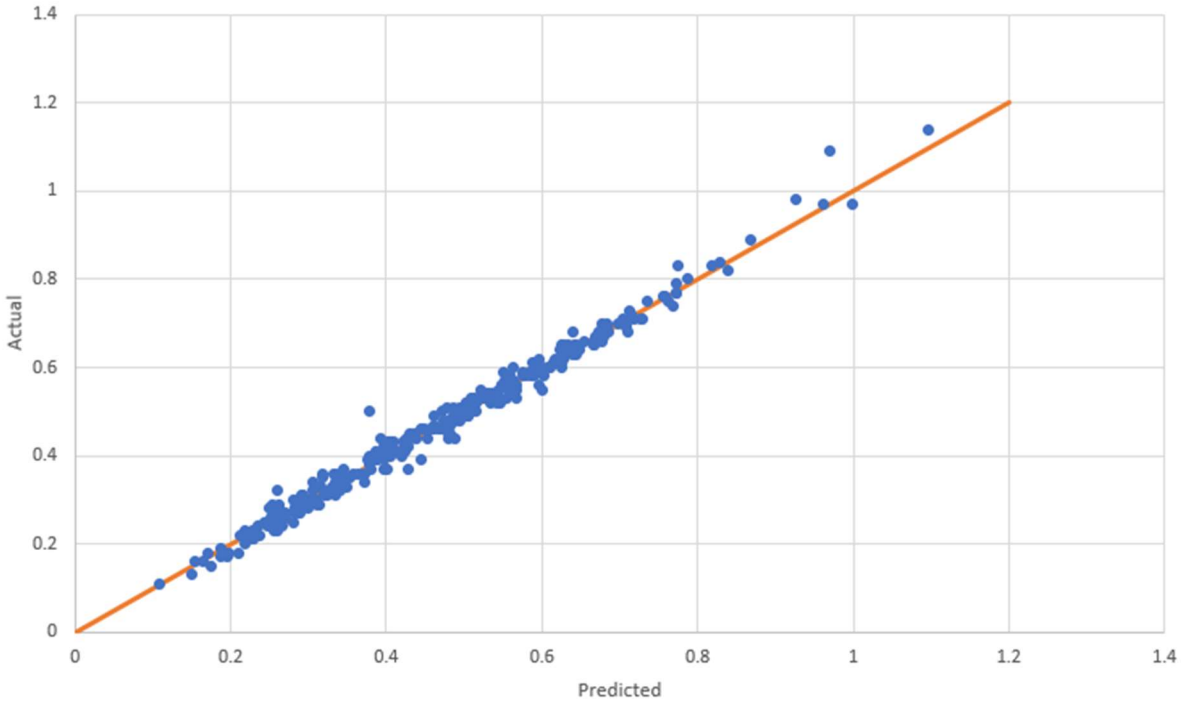
0.33488	0.34
0.330991	0.32
0.42796	0.37
0.473121	0.48
0.229686	0.21
0.372141	0.36
0.563469	0.56
0.248282	0.24
0.727099	0.71
0.227823	0.23
0.624661	0.61
0.961225	0.97
0.30913	0.3
0.711375	0.68
0.236637	0.22
0.439111	0.44
0.997879	0.97
0.597899	0.6
0.352944	0.35
0.755827	0.76
0.479598	0.45
0.164132	0.16
0.504708	0.49
0.405887	0.41
0.509522	0.53
0.829386	0.84
0.108355	0.11
0.482923	0.45
0.772301	0.79
0.290441	0.3
0.548881	0.56
0.266555	0.26
0.42059	0.4
0.485708	0.49
0.259662	0.23
0.676442	0.66
0.614602	0.61
0.400881	0.37
0.471187	0.46
0.52621	0.53
0.639694	0.64
0.379255	0.5
0.575616	0.58
0.254899	0.27

0.491591	0.49
0.620282	0.62
0.370249	0.36
0.708386	0.7
0.312969	0.29
0.712676	0.71
0.96985	1.09
0.640078	0.68
0.430332	0.44
0.345992	0.37
0.592414	0.6
0.515814	0.5
0.227646	0.22
0.404555	0.43
0.926799	0.98
0.287973	0.27
0.260784	0.28
0.529154	0.54
0.534254	0.53
0.729883	0.71
0.193786	0.18
0.38703	0.4
0.209225	0.18
0.194815	0.17
0.487368	0.5
0.772585	0.77
0.305533	0.34
1.097434	1.14
0.493579	0.51
0.55113	0.59
0.625986	0.65
0.339132	0.34
0.587192	0.61
0.536617	0.54
0.405131	0.4
0.253727	0.29
0.344737	0.35
0.444268	0.39
0.426474	0.44
0.678864	0.67
0.504915	0.5
0.30518	0.3
0.383087	0.39
0.385011	0.4

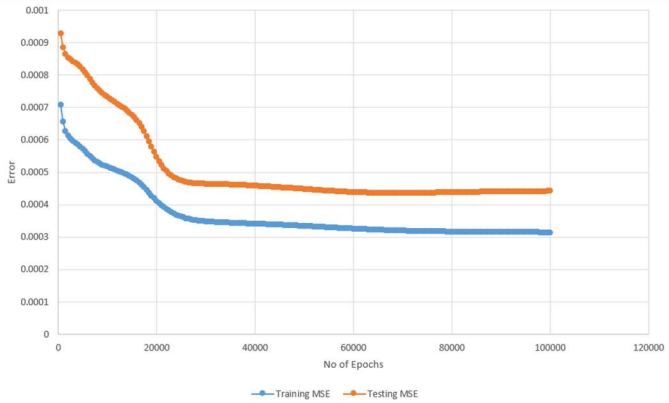
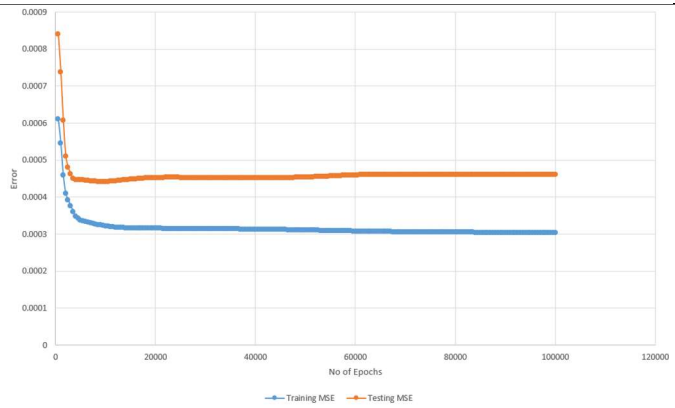
0.387235	0.39
0.249114	0.28
0.758975	0.76
0.219877	0.21
0.46735	0.46
0.29066	0.29
0.580261	0.58
0.282897	0.28
0.598188	0.59
0.601421	0.58
0.462066	0.46
0.461377	0.49
0.396446	0.39
0.45341	0.44
0.410272	0.43
0.235245	0.24
0.562433	0.56
0.51411	0.53
0.325301	0.31
0.316672	0.33
0.403072	0.43
0.423977	0.43
0.186425	0.19
0.478963	0.48
0.488528	0.44
0.150084	0.13
0.624357	0.6
0.684246	0.7
0.340637	0.32
0.187457	0.17
0.552309	0.57
0.599498	0.6
0.171185	0.18
0.666945	0.65
0.481674	0.47
0.666044	0.66
0.529852	0.54
0.67049	0.67
0.384939	0.4
0.397026	0.37
0.643866	0.63
0.534154	0.52
0.392452	0.44
0.422925	0.42

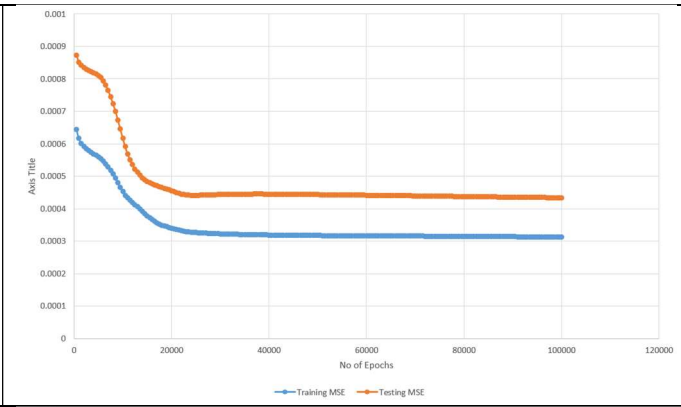
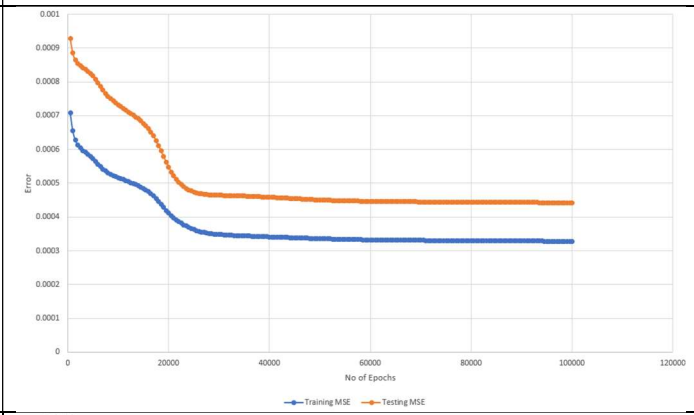
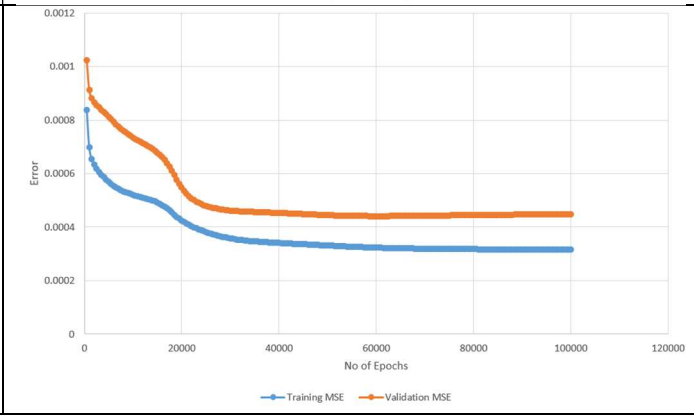
0.243491	0.25
0.259717	0.32
0.562174	0.55
0.423651	0.41
0.28406	0.27
0.616296	0.62
0.292583	0.31
0.707802	0.69
0.380727	0.37
0.479421	0.46
0.64299	0.65
0.639181	0.63
0.270465	0.26
0.198518	0.18
0.332954	0.36
0.285169	0.27
0.283364	0.28
0.478875	0.51
0.489167	0.48
0.818946	0.83
0.509149	0.51
0.563879	0.57
0.719115	0.71
0.607941	0.6
0.299545	0.28
0.420848	0.43
0.396234	0.42
0.502167	0.52
0.263251	0.27
0.372844	0.34
0.228195	0.21
0.462187	0.47
0.595785	0.56
0.362823	0.36
0.448313	0.46
0.627582	0.64
0.376566	0.39
0.609832	0.6
0.395317	0.4
0.644931	0.64
0.624384	0.63
0.254574	0.24
0.284732	0.28
0.544796	0.55

0.542838	0.52
0.505403	0.49
0.40232	0.41
0.281284	0.25
0.868975	0.89
0.254873	0.23
0.265261	0.24
0.280207	0.3
0.566874	0.53
0.218824	0.23
0.153128	0.16
0.600042	0.55
0.563675	0.6
0.474336	0.47
0.437405	0.45
0.519744	0.53
0.378061	0.38
0.378322	0.4
0.413278	0.41
0.531771	0.53
0.584974	0.58
0.388778	0.39
0.334634	0.31
0.697312	0.7
0.703261	0.71
0.457383	0.46
0.291576	0.3
0.528587	0.54
0.593992	0.59
0.775361	0.83
0.261897	0.25
0.320891	0.31
0.632773	0.65
0.669068	0.67
0.329871	0.33
0.513664	0.51



Final Evaluation

Name of Modification	Graph	No of Epochs to reach minima	Minima Testing MSE (Point just before it reached overtraining)
Basic		68,500	4.35959071181917E-4
Momentum		9500	4.4173397378744655E-4

Bold Driver		24,500	4.4137172538335595E-4
Annealing		100,000+	4.4230541128348116E-4
Weight Decay		63,500	4.41445957280988E-4

For training the algorithm most quickly, momentum is definitely the best model as it trains the algorithm to its fullest extent in 9,500 epochs

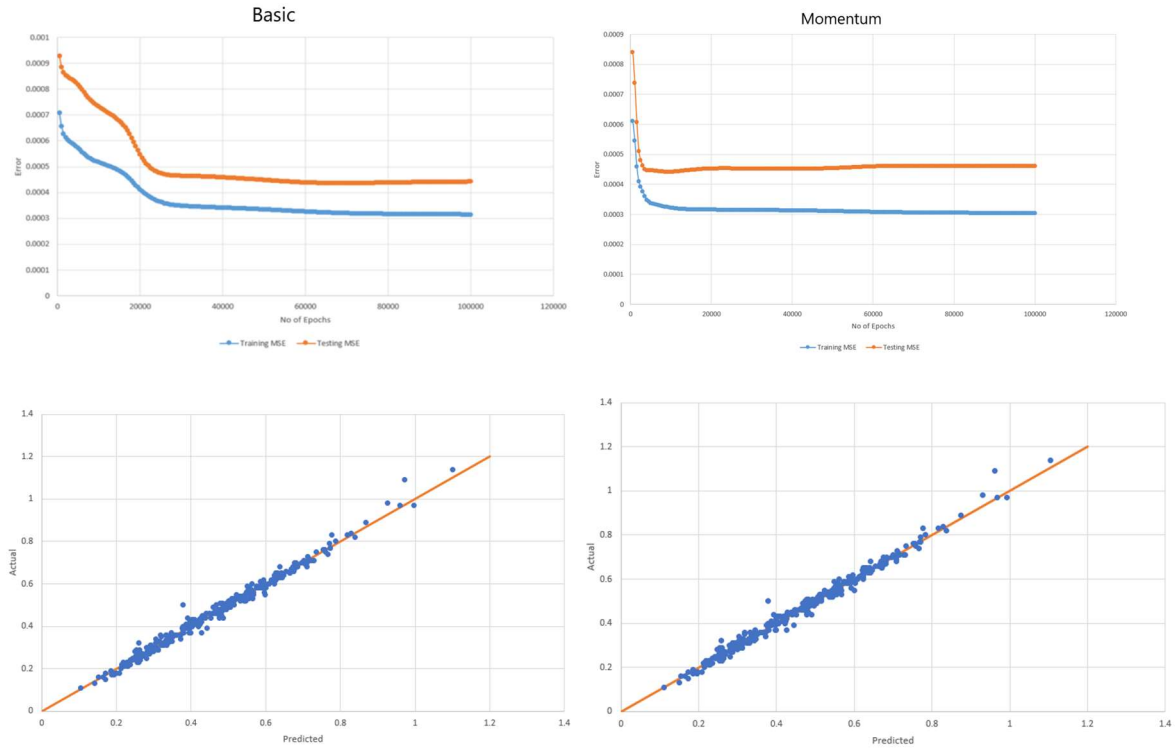
For reaching a more accurate rate of predicting values, Bold Driver is the best model, only by slightly as it has a slightly better MSE than the rest of the models.

However, it requires significantly more epochs to reach a slightly lower Mean Squared Error over Momentum

Though, each model improvement reached their respective minima in a smaller no of epochs than the basic model, with the exception of annealing, which didn't seem to make much improvement on reducing the number of epochs at all.

In conclusion, I would say that using Momentum, along with 4 inputs and 8 hidden nodes is the best model, as 4 inputs with 8 hidden nodes produced the lowest overall Mean Squared error, and

Momentum significantly reduces the required training time from the Basic model at a much greater amount over the rest of the models



Comparing the two models against 4 inputs and 8 hidden nodes (our best modifications), we can clearly tell that momentum trains the data faster due it its much steeper gradient, but also adapts to unseen data significantly faster than the Basic model, while still retaining approximately the same minima MSE. Meaning we get roughly the same prediction accuracy, but in a much shorter number of epochs.

Comparison to LINEST function

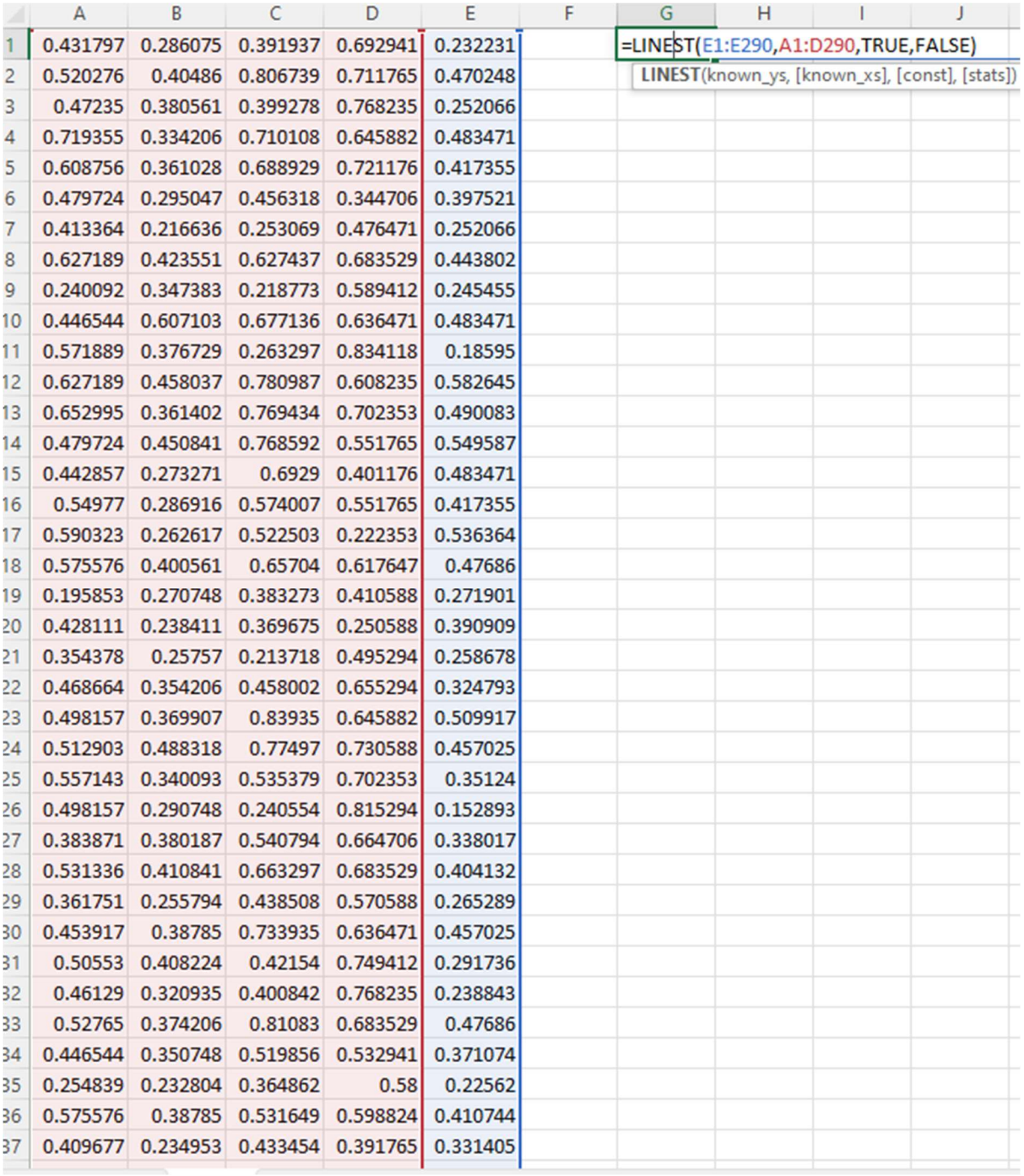
Now I have chosen Momentum as the most optimal Model, I will now compare against the LINEST approach.

4 Inputs

I will use the testing data set with 4 inputs, as that is what the majority of our MSE calculations were based on for our previous models.

I also made sure to use the Testing data set as that is the data set that I used for calculating the MSE for my Back Propagation algorithm

I use the LINEST function to calculate the weights and the biases



	A	B	C	D	E	F	G	H	I	J
1	0.431797	0.286075	0.391937	0.692941	0.232231		=LINEST(E1:E290,A1:D290,TRUE,FALSE)			
2	0.520276	0.40486	0.806739	0.711765	0.470248		LINEST(known_ys, [known_xs], [const], [stats])			
3	0.47235	0.380561	0.399278	0.768235	0.252066					
4	0.719355	0.334206	0.710108	0.645882	0.483471					
5	0.608756	0.361028	0.688929	0.721176	0.417355					
6	0.479724	0.295047	0.456318	0.344706	0.397521					
7	0.413364	0.216636	0.253069	0.476471	0.252066					
8	0.627189	0.423551	0.627437	0.683529	0.443802					
9	0.240092	0.347383	0.218773	0.589412	0.245455					
10	0.446544	0.607103	0.677136	0.636471	0.483471					
11	0.571889	0.376729	0.263297	0.834118	0.18595					
12	0.627189	0.458037	0.780987	0.608235	0.582645					
13	0.652995	0.361402	0.769434	0.702353	0.490083					
14	0.479724	0.450841	0.768592	0.551765	0.549587					
15	0.442857	0.273271	0.6929	0.401176	0.483471					
16	0.54977	0.286916	0.574007	0.551765	0.417355					
17	0.590323	0.262617	0.522503	0.222353	0.536364					
18	0.575576	0.400561	0.65704	0.617647	0.47686					
19	0.195853	0.270748	0.383273	0.410588	0.271901					
20	0.428111	0.238411	0.369675	0.250588	0.390909					
21	0.354378	0.25757	0.213718	0.495294	0.258678					
22	0.468664	0.354206	0.458002	0.655294	0.324793					
23	0.498157	0.369907	0.83935	0.645882	0.509917					
24	0.512903	0.488318	0.77497	0.730588	0.457025					
25	0.557143	0.340093	0.535379	0.702353	0.35124					
26	0.498157	0.290748	0.240554	0.815294	0.152893					
27	0.383871	0.380187	0.540794	0.664706	0.338017					
28	0.531336	0.410841	0.663297	0.683529	0.404132					
29	0.361751	0.255794	0.438508	0.570588	0.265289					
30	0.453917	0.38785	0.733935	0.636471	0.457025					
31	0.50553	0.408224	0.42154	0.749412	0.291736					
32	0.46129	0.320935	0.400842	0.768235	0.238843					
33	0.52765	0.374206	0.81083	0.683529	0.47686					
34	0.446544	0.350748	0.519856	0.532941	0.371074					
35	0.254839	0.232804	0.364862	0.58	0.22562					
36	0.575576	0.38785	0.531649	0.598824	0.410744					
37	0.409677	0.234953	0.433454	0.391765	0.331405					

	G	H	I	J	K
	-0.45584	0.380802	0.4054	0.311213	0.155398

I can then delete them from the excel file and store them inside the java class

```
public class LINEST {
    private ArrayList<ArrayList<Double>> data = BackPropagation.testingData;
    private double[] weights = new double[5];
    private double[] predictedOutputs = new double[data.size()-1];
    private double[] actualOutputs = new double[data.size()-1];
    private double mse=0;

    public LINEST(){
        this.weights[0]=-0.45583798;
        this.weights[1]=0.380802074;
        this.weights[2]=0.405399896;
        this.weights[3]=0.311213449;
        this.weights[4]=0.155398408;
    }
```

I also set up the test data ArrayList, the predicted outputs and the actual outputs, along with the mse variable

```
public void run(){
    double total=0;
    for (int i=0; i<data.size(); i++) {
        double[] inputs = new double[data.get(0).size()-1];
        System.out.println(inputs.length);
        for (int j = 0; j < inputs.length; j++) {
            System.out.println(j);
            inputs[j]=data.get(i).get(j);
        }
        actualOutputs[i]=deStandardiseOutput(data.get(i).get(data.get(i).size()-1));
        predictedOutputs[i]=deStandardiseOutput(calculate(inputs));
    }
    for (int i = 0; i < data.size(); i++) {
        total+=Math.pow((predictedOutputs[i]-actualOutputs[i]),2);
    }
    mse=total/(data.size()-1);
    plotPredVsActual();
}
```

```
public double deStandardiseOutput(double value){
    return (((value-0.1)/0.8)*(1.28-0.07)+0.07);
}
```

```
public void plotPredVsActual(){
    //plot data
    double[][] predVsActual = new double[data.size()][2];
    for (int i = 0; i < data.size(); i++) {
        predVsActual[i][0] = predictedOutputs[i];
        predVsActual[i][1] = actualOutputs[i];
    }
    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet sheet = workbook.createSheet("Data");

    int rowNum = 0;
    for (double[] rowData : predVsActual) {
      Row row = sheet.createRow(rowNum++);

      int colNum = 0;
      for (double field : rowData) {
        Cell cell = row.createCell(colNum++);
        cell.setCellValue(field);
      }
    }
    try (FileOutputStream outputStream = new FileOutputStream("PredVsActual.xlsx")) {
        workbook.write(outputStream);
    }catch (IOException e) {
        e.printStackTrace();
      }
}
```

```java
public double calculate(double[] inputs){
    double dp = 0;
    for (int i = 0; i < inputs.length; i++) {
        dp+=inputs[i]*weights[i];
    }
    dp+=weights[weights.length-1]; //plus the bias term
    return dp;
}
```

I go through each row, store the actual outputs for each row, and calculate the predicted outputs by doing a dot product on the row, along with the column of weights, and I add the single bias at the end

I sum all the square differences into a total, and divide by the number of rows in the data (290)

```java
public void printMSE(){
    System.out.println("MSE: "+mse);
}
```

And I can print the MSE using this method defined also in the LINEST class

```java
LINEST linest = new LINEST();
linest.run();
linest.printMSE();
```

Running this Model gives me a MSE of 0.0677585235099156, which is significantly higher than any of my other models.

However, given its very fast run time and easy setup, it would be a good solution for simpler, more linear problems, with less parameters.