CPEN502 Assignment1

Part 1a - Backpropagation Learning

Chenchen Zhu student#:49613201

BINARY representation

Number of trails	learning rate	momentum	Average epochs	Max epochs	Min epochs
500	0.2	0	3840	9356	2175
2000	0.2	0	3766	9373	1859
500	0.2	0.9	406	933	210
2000	0.2	0.9	400	1220	208

BIPOLAR representation

Number of trails	learning rate	momentum	Average epochs	Max epochs	Min epochs
500	0.2	0	272	595	189
2000	0.2	0	273	557	178
500	0.2	0.9	30	60	17
2000	0.2	0.9	30	69	16

Example Graphs:

(a) binary representation

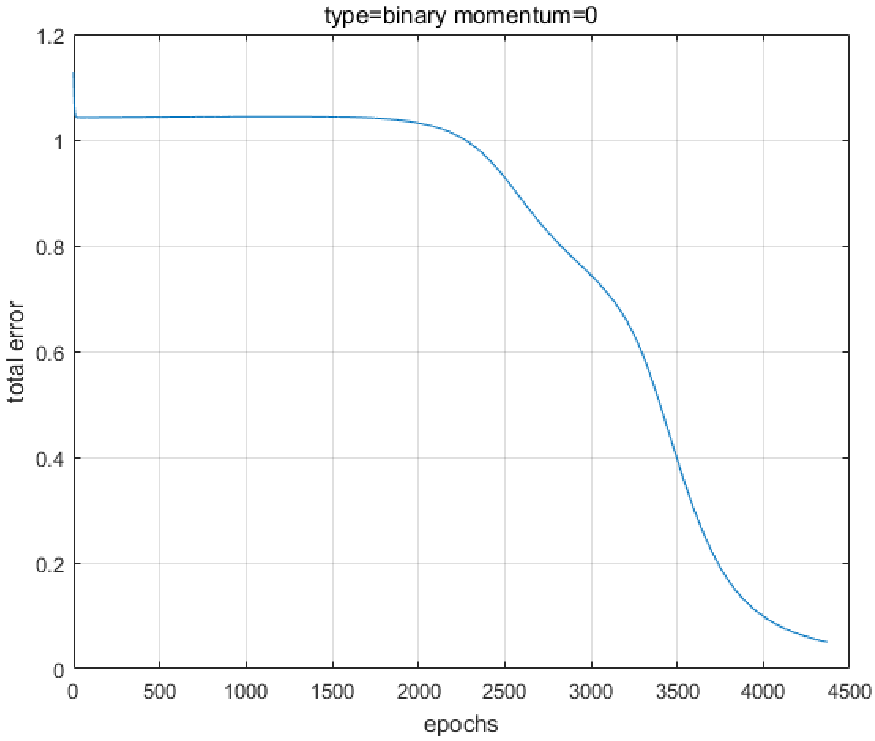

Figure a-1) binary representation and momentum=0

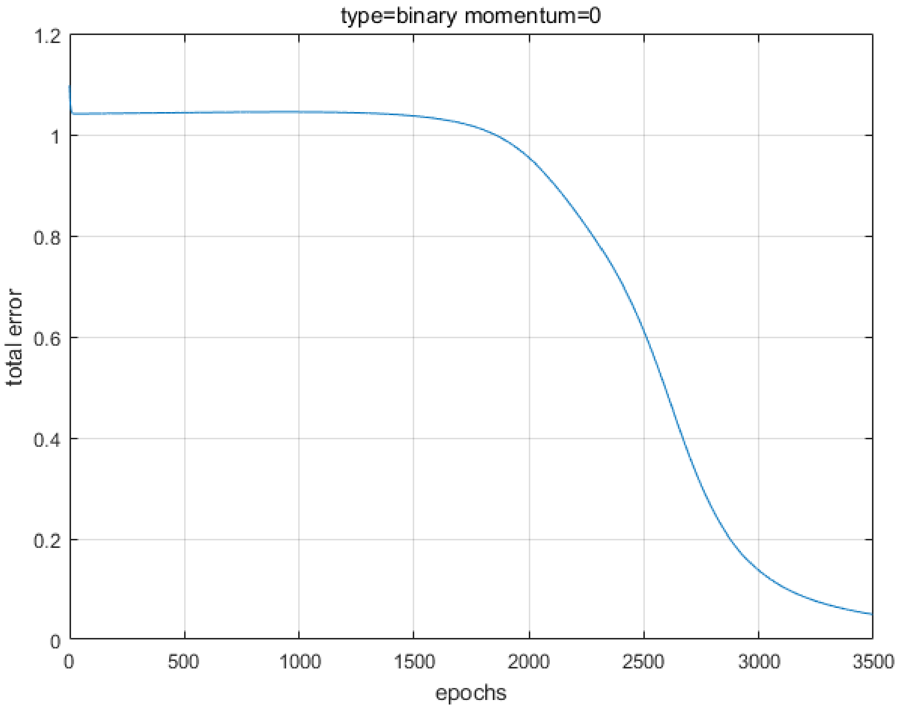

Figure a-2) binary representation and momentum=0

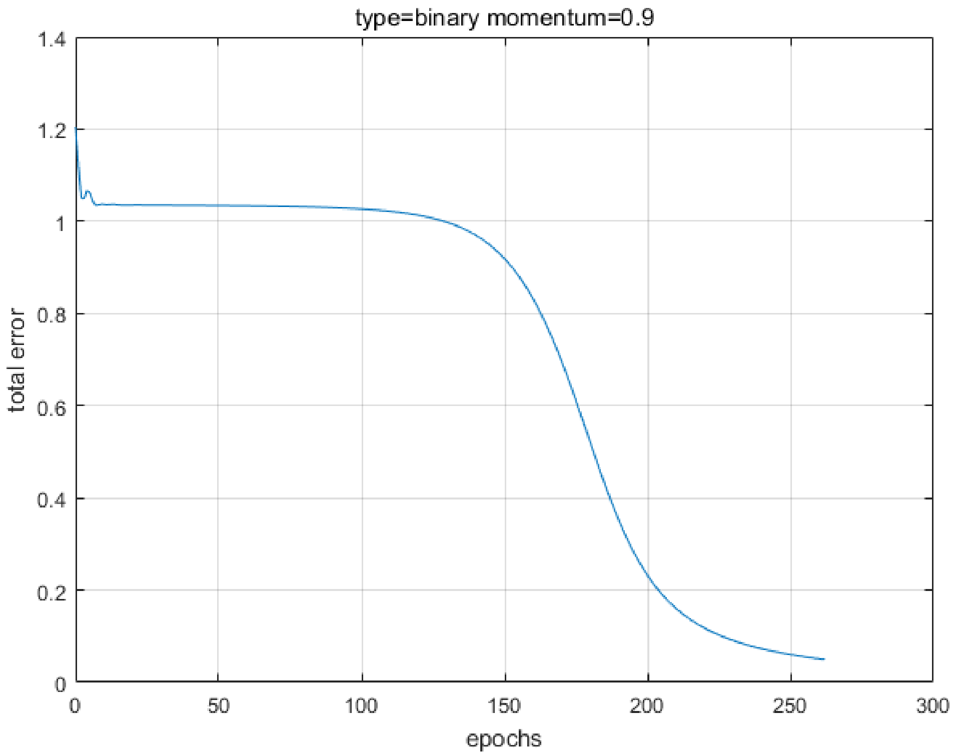

Figure a-3) binary representation and momentum=0.9

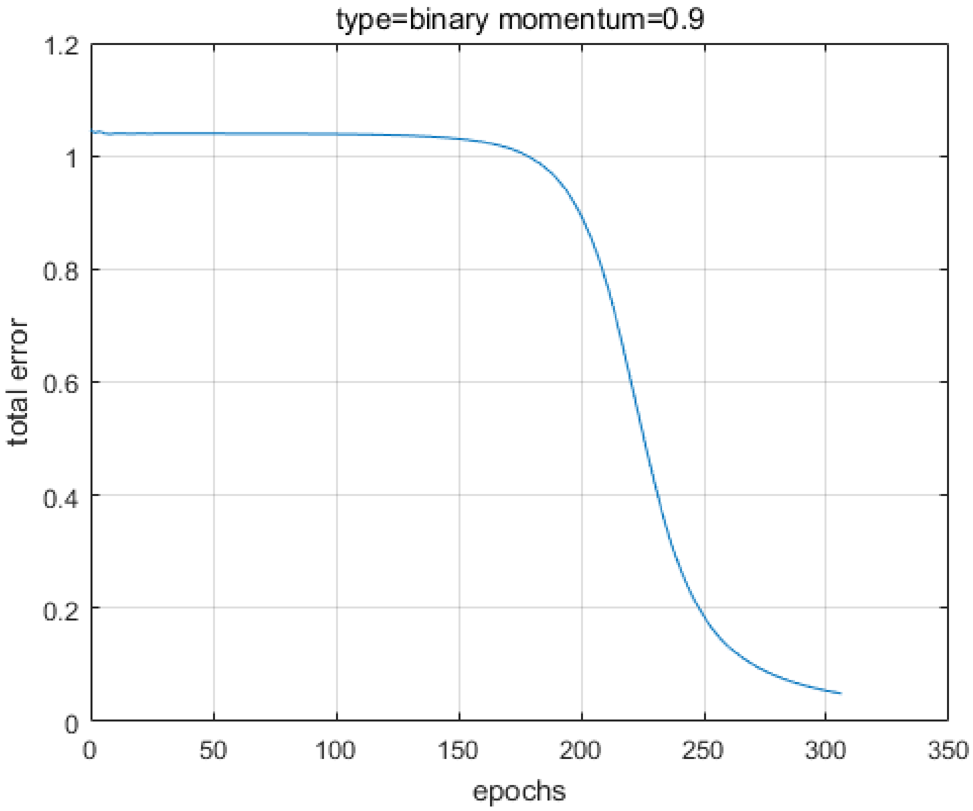

Figure a-4) binary representation and momentum=0.9

(b) bipolar representation

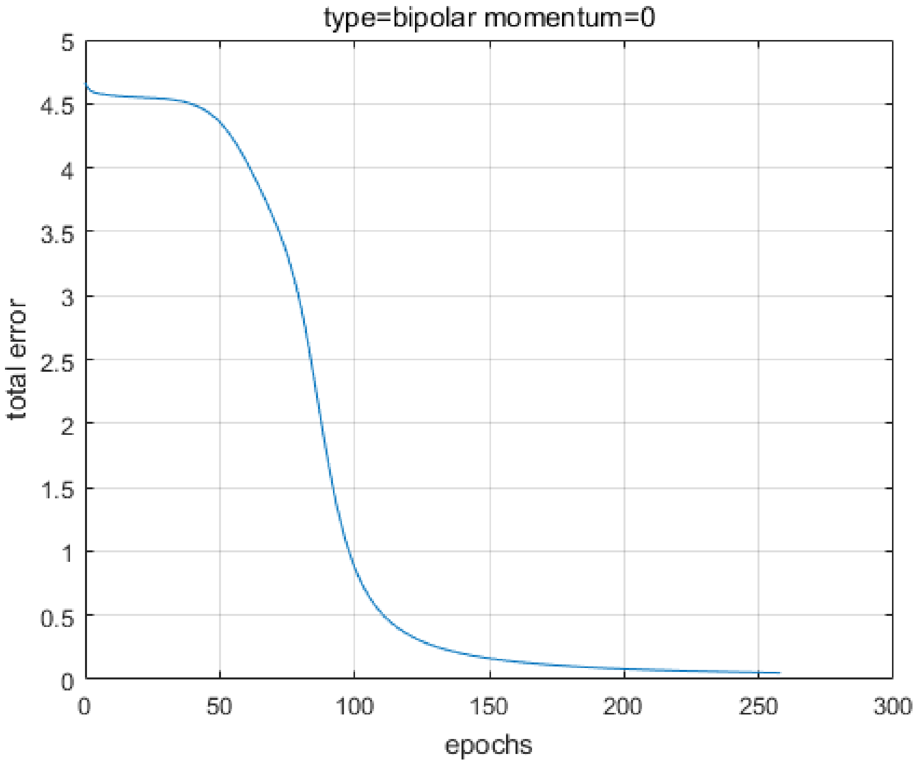

Figure b-1) bipolar representation and momentum=0

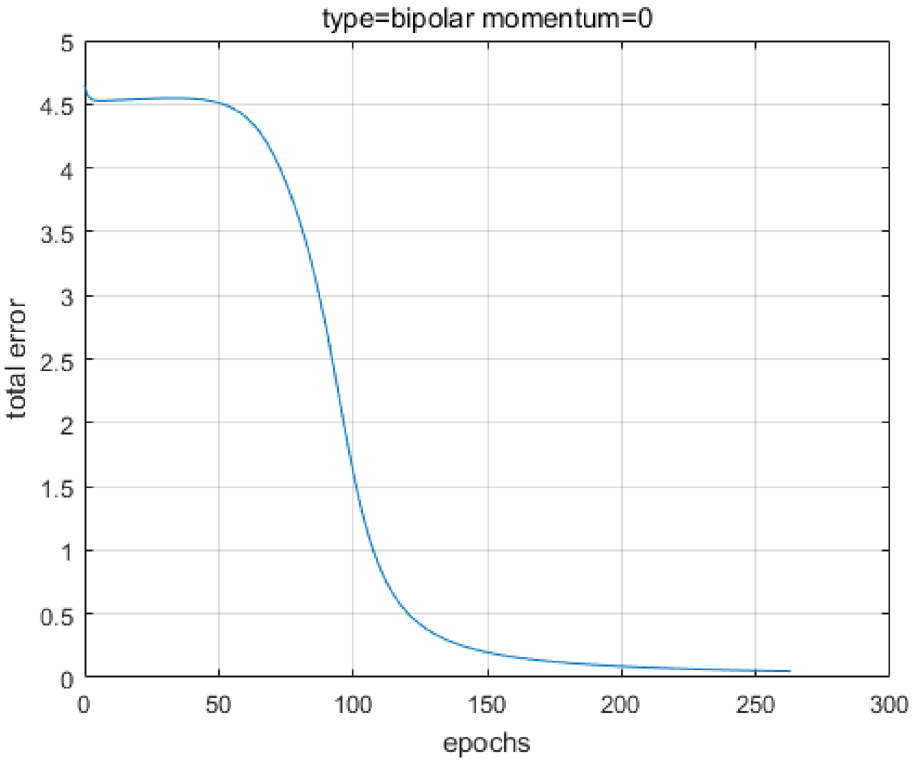

Figure b-2) bipolar representation and momentum=0

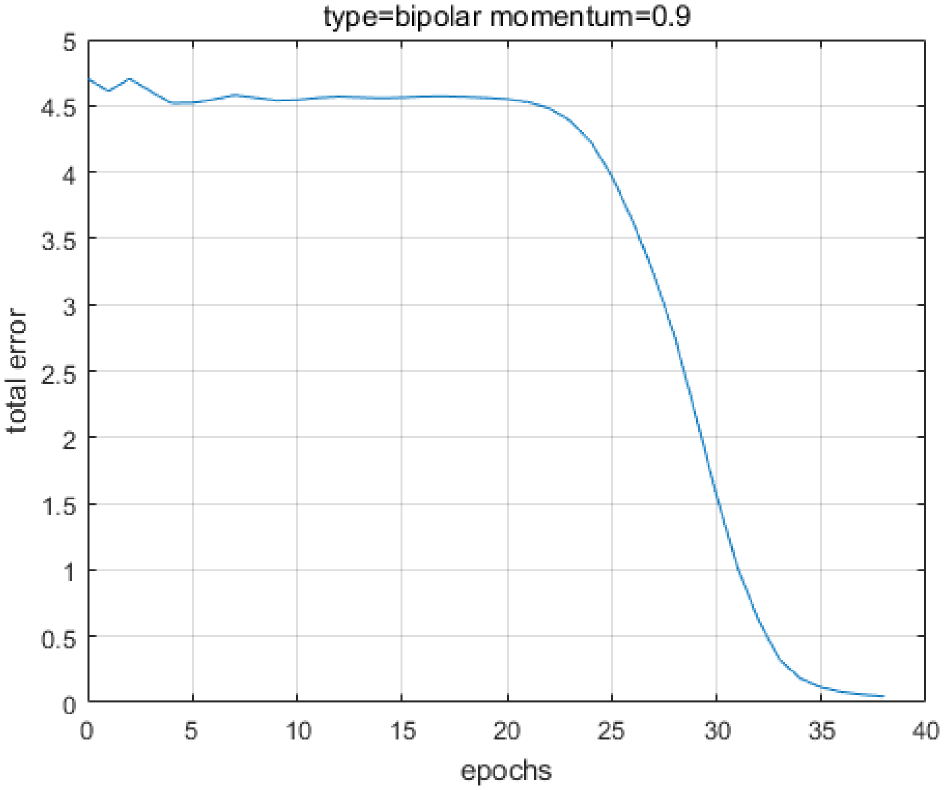

Figure b-3) bipolar representation and momentum=0

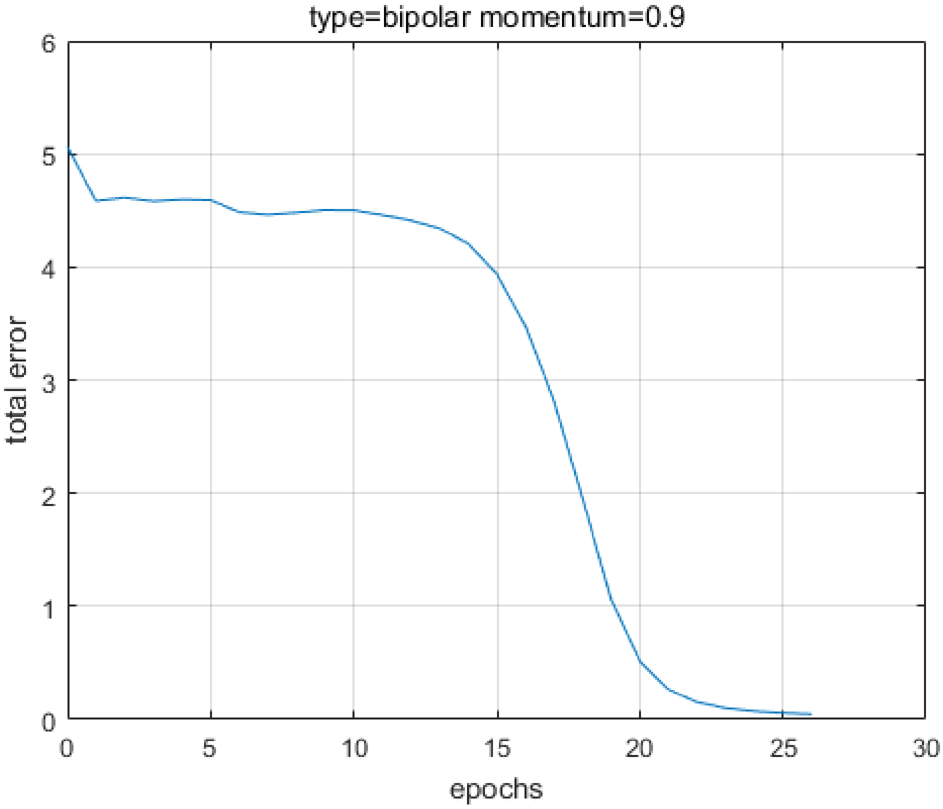

Figure b-4) bipolar representation and momentum=0

Appendix

NeuralNet.java

```
package Assignment1;

import java.io.Console;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

import org.omg.CORBA.PRIVATE_MEMBER;
import org.omg.CORBA.PUBLIC_MEMBER;

import Sarb.NeuralNetInterface;

public class NeuralNet implements NeuralNetInterface {

    static double bias = 1.0;
    private int argNumInputs;
    private int argNumHidden;
    private int argNumOutputs;
    private int argNumTrainingSet;
    private double argLearningRate;
    private double argMomentumTerm;
    private double argA;
    private double argB;


    private ArrayList<Neuron> inputLayer = new
ArrayList<Neuron>();
    private ArrayList<Neuron> hiddenLayer = new
ArrayList<Neuron>();
    private ArrayList<Neuron> outputLayer = new
ArrayList<Neuron>();
    private ArrayList<ArrayList<Neuron>> allLayerArrayList =
new ArrayList<ArrayList<Neuron>>();
    private ArrayList<Double> totalErrorInEachEpoch = new
ArrayList<Double>(); //save the total error in each epoch
    public int totalEpochNum=1;
    private Neuron biasNeuron = new Neuron("bias",0,1);

    public void allLayers() {
       allLayerArrayList.add(hiddenLayer);
       allLayerArrayList.add(outputLayer);
    }
    public NeuralNet(int argNumInputs, int argNumHidden, int
argNumOutputs, int argNumTrainingSet, double argLearningRate,
double argMomentumTerm, double argA,
         double argB) {
      this.argNumInputs = argNumInputs;
      this.argNumHidden = argNumHidden;
      this.argNumOutputs = argNumOutputs;
      this.argNumTrainingSet = argNumTrainingSet;
      this.argLearningRate = argLearningRate;
      this.argMomentumTerm = argMomentumTerm;
      this.argA = argA;
      this.argB = argB;

      //this.initializeTrainingSet();


   }

    public void buildLayers() {
       //build input layer
       for(int i=0; i<argNumInputs; i++) {
          String id = "inputLayerNeuron" +
Integer.toString(i);
          Neuron e = new Neuron(id,argA,argB);
          inputLayer.add(e);
       }
       //build hidden layer
       for(int i=0; i<argNumHidden; i++) {
          String id = "hiddenLayerNeuron"
+Integer.toString(i);
          Neuron e = new
Neuron(id,"customSigmoid",inputLayer,biasNeuron,argA,argB);
          hiddenLayer.add(e);
       }
       //build output layer
       for(int i=0; i<argNumOutputs; i++) {
          String id = "outputLayerNeuron"
+Integer.toString(i);
```

```
          Neuron e = new
Neuron(id,"customSigmoid",hiddenLayer,biasNeuron,argA,argB);
          outputLayer.add(e);
       }
       biasNeuron.setNeuronOut(1.0);
    }

    public double getWeightRandom(double lowerbound, double
upperbound) {
       Random random = new Random();
       double weight = random.nextDouble() * (upperbound-
lowerbound) + lowerbound;
       return weight;
    }

    public void initializeWeights() {
       double lowerbound = -0.5;
       double upperbound = 0.5;
       for(ArrayList<Neuron> al: allLayerArrayList) {
          for(Neuron neuron: al) {
          ArrayList<Edge> edges = neuron.getInEdges();
          for(Edge currentedge: edges) {

currentedge.setWeight(getWeightRandom(lowerbound,upperbound));
          }
          Edge edge = neuron.getBiasEdge();

edge.setWeight(getWeightRandom(lowerbound,upperbound));

       }

      }
    }


   public double sigmoid(double x) {
       return 0;
    }

    public double customSigmoid(double x) {
       return 0;
    }
```

```
public void zeroWeigths() {
   for(ArrayList<Neuron> al: allLayerArrayList) {
      for(Neuron neuron: al) {
     ArrayList <Edge> inEdges = neuron.getInEdges();
     for(Edge e: inEdges) {
        e.setWeight(0);
     }
  }
 }
}

public double[] outputFor(double [] x) {
  //setInputData(X);
  // System.out.println(Arrays.deepToString(X));
   //System.out.println(Arrays.toString(x));
   for(int i=0; i< inputLayer.size(); i++) {
      inputLayer.get(i).setNeuronOut(x[i+1]);
   }
   forwardPropagate();
   double  outputs[] = getOutputs();
   return outputs;
}

public double[] getOutputs() {
  double [] outputs = new double[outputLayer.size()];
  //System.out.println(outputLayer.size());
  for(int i = 0; i < outputLayer.size(); i++) {
     outputs[i] =outputLayer.get(i).getNeuronout();
  }
  return outputs;
}

public void forwardPropagate() {
  for(ArrayList<Neuron> al: allLayerArrayList) {
     for(Neuron n: al) {
     n.forwardPropagate();
  }
   }
}

public void backwardPropagate(double output[]) {
  //int i = 0;             //?
  for(Neuron n : outputLayer) {
         double y = n.getNeuronout();
         double z = output[0];
         ArrayList<Edge> edges = n.getInEdges();

         for(Edge e : edges) {
            double x = e.getInputValue();
            double error = customSigmoidDerivative(y)*(z-y);
            e.setError(error);
            double delta =argMomentumTerm*e.getDelta() +
argLearningRate*error*x; //current link's deltaweight has not
be updated yet, so it is previous delta w
            double newWeight = e.getWeight() + delta;

            e.setDelta(delta);
            e.setWeight(newWeight);
         }
         //i++;
      }
      //System.out.println("hey");
      for(Neuron n: hiddenLayer) {   //different way to
calculate error for nodes in hidden layer
         double y =n.getNeuronout();
         ArrayList<Edge> edges = n.getInEdges();
         //System.out.println(edges.size());
         for(Edge e : edges) {
            double x = e.getInputValue();
            double sumWeightedError = 0;
            for(Neuron outNeuron: outputLayer) {
               //System.out.println(edges.size());
               double whj =
outNeuron.getInEdgeMap(n.getNeuronId()).getWeight();

               double errorh =
outNeuron.getInEdgeMap(n.getNeuronId()).getError();
               sumWeightedError = sumWeightedError + whj
*errorh;
            }
            double error =
customSigmoidDerivative(y)*sumWeightedError;
            e.setError(error);
            double delta =argMomentumTerm * e.getDelta() +
argLearningRate*error*x;
            double newWeight = e.getWeight() + delta;
            e.setDelta(delta);
            e.setWeight(newWeight);
         }
      }
    }

    public double train(double[][] X, double[][] Y){  //one
epoch
       double totalError = 0;
       for(int i=0; i<X.length; i++) {
          double error = 0;
          double outputZ[] = outputFor(X[i]);
         // System.out.println(Arrays.deepToString(X));
         // System.out.println(Arrays.toString(outputZ));
         // System.out.println(outputZ[0]);
          for(int j = 0; j<argNumOutputs; j++) {
             error = error + Math.pow(outputZ[j]-Y[i][j], 2);
          }
          this.backwardPropagate(Y[i]);
          totalError = totalError + error;

       }
        totalErrorInEachEpoch.add(totalError);

       return totalError;
    }

    public double train(double [] x, double argValue) {
       return 0;
    }

    public void runNeuralNet(double errorThreshold,double[][]
X, double[][] Y) {
      int step = 1;
      double error;
      error = train(X,Y);
      //System.out.print(error);
      while(error > errorThreshold) {
         error = train(X,Y);
         step++;
         totalEpochNum++;
      }
      System.out.println("Total error in the last epoch is " +
error + "\n");
      System.out.println("Total number of epoches "+
totalEpochNum + "\n");
    }

   public ArrayList<Double> getErrorArray(){
      return this.totalErrorInEachEpoch;
   }

    public void save(File argFile) {

    }

    public void load(String argFileName) throws IOException{

    }

    public double customSigmoidDerivative(double y) {
       double result;
       if(argA==-1) {
          result=1.0/2.0 * (1-y) * (1+y);
       }
       else {
          result=y*(1-y);
       }
       return result;
    }

   public void printRunResults(ArrayList<Double> errors, 
String fileName) throws IOException {
      int epoch;
      PrintWriter printWriter = new PrintWriter(new 
FileWriter(fileName));
      printWriter.printf("Epoch Number, Total Squared Error, 
\n");
      for(epoch = 0; epoch < errors.size(); epoch++) {
         printWriter.printf("%d, %f, \n", epoch, 
errors.get(epoch));
      }
      System.out.print("success!");
      printWriter.flush();
      printWriter.close();
   }
}
```

Neuron.java

```java
package Assignment1;

import java.io.PipedInputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class Neuron {
    private String neuronId;
    private String activationFunction;
    private ArrayList <Edge> inEdges = new ArrayList <Edge>();
    private HashMap<String, Edge> allInEdges = new 
HashMap<String,Edge>();
    public double NeuronOut = 0; //neuron's value
    private double a;
    private double b;
     private Edge biasEdge;
    final double bias = 1;



    //Constuctor for input layer neurons
    public Neuron(String id,double a,double b) {
        this.neuronId = id;
        this.a = a;
        this.b = b;
    }
    // Constructor for hidden,output layer neurons
    public Neuron(String id, String activationFunction, 
List<Neuron> inNeurons, Neuron bias,double a,double b) {
        this.neuronId = id;
        this.activationFunction = activationFunction;
        this.a = a;
        this.b = b;
//      setActivationFunction(activationFunction);
        addInputEdges(inNeurons);
        addBiasInput(bias);
    }
     public Edge getBiasEdge() {
        return this.biasEdge;
     }
    public double getNeuronout() {
        return this.NeuronOut;
    }

    public void setNeuronOut(double out) {
        this.NeuronOut = out;
    }

    public String getNeuronId() {
        return this.neuronId;
    }

    public String getActivationFunction() {
        return this.activationFunction;
    }

    public ArrayList<Edge> getInEdges(){
         return this.inEdges;
     }

    public Edge getInEdgeMap(String neuronId) {
        return allInEdges.get(neuronId);
    }

//    public void setActivationFunction(String
activationFunction) {
//
//    }

    public void addInputEdges(List<Neuron> inNeurons) {
        for(Neuron neuron: inNeurons) {
            Edge edge = new Edge(neuron,this);
            inEdges.add(edge);
            allInEdges.put(neuron.getNeuronId(), edge);
        }
    }
    public void addBiasInput(Neuron bias) {
        Edge edge = new Edge(bias, this);
        inEdges.add(edge);
        this.biasEdge = edge;
        allInEdges.put(bias.getNeuronId(), edge);
    }

    public void forwardPropagate() {
        double weightedSum = calculateWeightedSum(inEdges);
        this.NeuronOut = customSigmoid(weightedSum);
    }

    public double calculateWeightedSum(ArrayList<Edge> inEdges)
{
        double sum = 0;
        for (Edge e: inEdges){
            double weight = e.getWeight();
            double value = e.getInputValue();
            sum = sum + weight*value;
        }

        if (biasEdge != null) {
            sum = sum + (this.biasEdge.getWeight()*this.bias);
        }
        return sum;
    }

    public double sigmoid(double weightedSum) {
        return 2/(1 + Math.exp(-weightedSum))-1;
    }
    public double customSigmoid(double weightedSum) {
        return (b-a)/(1+Math.exp(-weightedSum))+a;
    }

}
```

Edge.java

```
package Assignment1;

import java.util.ArrayList;

public class Edge {
    private double weight = 0;
    private Neuron pre;
    private Neuron next;
    private double inputValue = 0;
    private double error = 0;
    private double delta = 0; //

     public Edge(Neuron pre, Neuron next) {
```

```java
        this.pre = pre;
        this.next = next;
    }
    public void setWeight(double weight) {
        this.weight = weight;
    }

    public void setDelta(double delta) {
        this.delta = delta;
    }

    public double getWeight() {
        return this.weight;
    }

    public double getDelta() {
        return this.delta;
    }

    public double getError() {
        return this.error;
    }

    public Neuron getPre() {
        return this.pre;
    }

    public Neuron getNext() {
        return this.next;
    }

    public double getInputValue() {
        inputValue = pre.getNeuronout();
        return inputValue;
    }

    public void setError(double error) {
        this.error = error;
    }
}
```

Test.java

```
package Assignment1;

import java.io.IOException;
import java.util.Arrays;

public class Test {
    private int argNumInputs = 2;
    private int argNumHidden = 4;
    private int argNumOutputs = 1;
    private int argNumTrainingSet = 4;
    private double argLearningRate = 0.2;
    private double argMomentumTerm = 0.9;
    private double bias = 1;
    private boolean binary = false;
    private double argA;
    private double argB;
    private double errorThreshold = 0.05;

    private double[][] inputX = new
double[argNumTrainingSet][argNumInputs+1]; //plus one bias
value
   private double[][] outputY = new
double[argNumTrainingSet][argNumOutputs];

   public void initializeTrainingSet() {
    if(binary) {
       argA = 0;
       argB = 1;
       inputX[0][0]=bias;
       inputX[0][1]=0;
       inputX[0][2]=0;

       inputX[1][0]=bias;
       inputX[1][1]=0;
       inputX[1][2]=1;

       inputX[2][0]=bias;
       inputX[2][1]=1;
       inputX[2][2]=0;

       inputX[3][0]=bias;
       inputX[3][1]=1;
       inputX[3][2]=1;
```

```
        outputY[0][0]=0;
        outputY[1][0]=1;
        outputY[2][0]=1;
        outputY[3][0]=0;
    }else {
        argA = -1;
        argB = 1;
        inputX[0][0]=bias;
        inputX[0][1]=-1;
        inputX[0][2]=-1;

        inputX[1][0]=bias;
        inputX[1][1]=-1;
        inputX[1][2]=1;

        inputX[2][0]=bias;
        inputX[2][1]=1;
        inputX[2][2]=-1;

        inputX[3][0]=bias;
        inputX[3][1]=1;
        inputX[3][2]=1;

        outputY[0][0]=-1;
        outputY[1][0]=1;
        outputY[2][0]=1;
        outputY[3][0]=-1;
    }
    }

   public void runNeuralNet() throws IOException {
        int aveEpochNum=0;
        int trials=500;
        int maxEpochNum=0;
        int minEpochNum=10000;
      for(int i=0;i<trials;i++) {
        initializeTrainingSet();
        // System.out.println(Arrays.deepToString(inputX));
        NeuralNet testNeuronNet = new
NeuralNet(argNumInputs,argNumHidden,argNumOutputs,argNumTraini
ngSet,argLearningRate,argMomentumTerm,argA,argB);
        testNeuronNet.buildLayers();
        testNeuronNet.allLayers();
        testNeuronNet.initializeWeights();
testNeuronNet.runNeuralNet(errorThreshold,inputX,outputY);
           if(testNeuronNet.totalEpochNum>maxEpochNum) {
             maxEpochNum = testNeuronNet.totalEpochNum;
           }
           if(testNeuronNet.totalEpochNum<minEpochNum) {
             minEpochNum = testNeuronNet.totalEpochNum;
           }
           aveEpochNum=aveEpochNum+testNeuronNet.totalEpochNum;

testNeuronNet.printRunResults(testNeuronNet.getErrorArray(),"F
://502result//bipolar-0.9//result"+i+".csv");
       }
       aveEpochNum = aveEpochNum/trials;
       System.out.println("ave:"+aveEpochNum);
       System.out.println("max:"+maxEpochNum);
       System.out.println("min:"+minEpochNum);

   }

   public static void main(String[] args) throws IOException {
       Test test = new Test();
       test.runNeuralNet();
    }

}
```