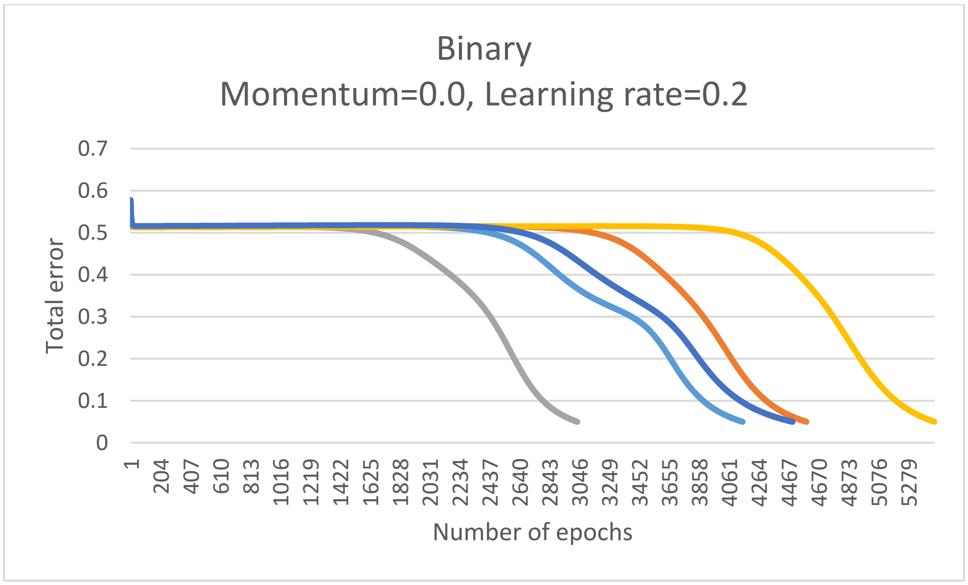
Project 1_a BP Network

98552169 Zhen Wang

1) Set up your network in a 2-input, 4-hidden and 1-output configuration. Apply the XOR training set. Initialize weights to random values in the range -0.5 to +0.5 and set the learning rate to 0.2 with momentum at 0.0.

a) Define your XOR problem using a binary representation. Draw a graph of total error against number of epochs. On average, how many epochs does it take to reach a total error of less than 0.05? You should perform many trials to get your results, although you don't need to plot them all.

The graph below shows the conditions of five trials, they use a binary representation with momentum set to be 0 and learning rate set to be 0.2.



Graph a. Binary representation with momentum 0 and learning rate 0.2

By implementing 10,000 times of trials, the statistics are achieved as below:

The average number to converge: 4001

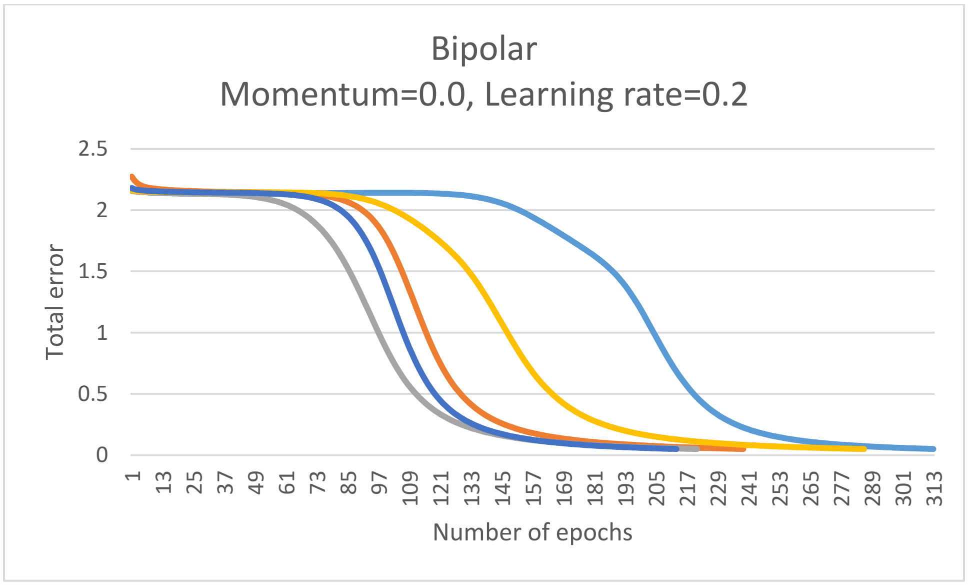
The maximum number to converge: 9879

The minimum number to converge: 2014

The convergence rate: 100%

b) This time use a bipolar representation. Again, graph your results to show the total error varying against number of epochs. On average, how many epochs to reach a total error of less than 0.05?

The graph below shows the conditions of five trials, they use a bipolar representation with momentum set to be 0 and learning rate set to be 0.2.



Graph b. Bipolar representation with momentum 0 and learning rate 0.2

By implementing 10,000 times of trials, the statistics are achieved as below:

The average number to converge: 254

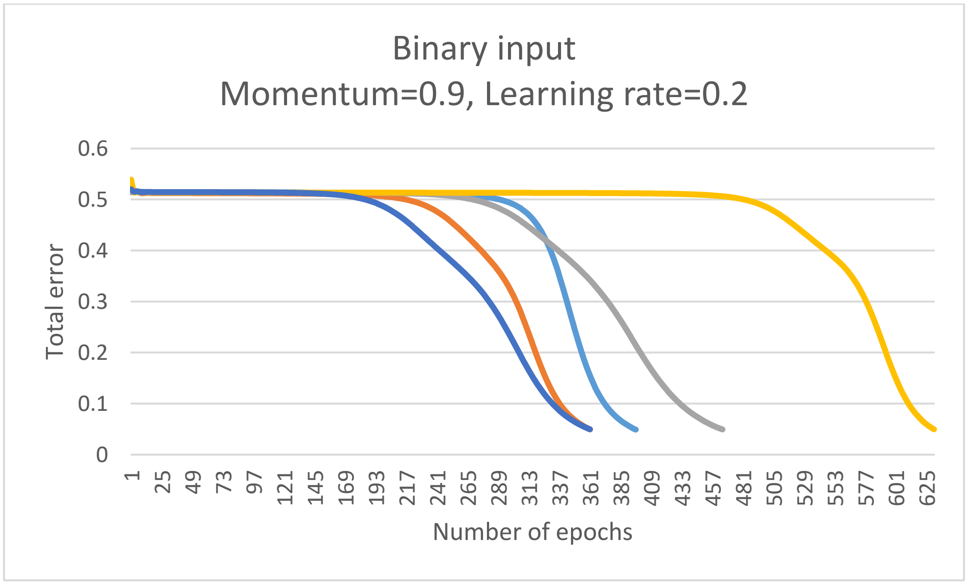
The maximum number to converge: 508

The minimum number to converge: 140

The convergence rate: 100%

c) Now set the momentum to 0.9. What does the graph look like now and how fast can 0.05 be reached?

c.1) The graph below shows the conditions of five trials, they use a binary representation with momentum set to be 0.9 and learning rate set to be 0.2.



Graph c1. Binary representation with momentum 0.9 and learning rate 0.2

By implementing 10,000 times of trials, the statistics are achieved as below:

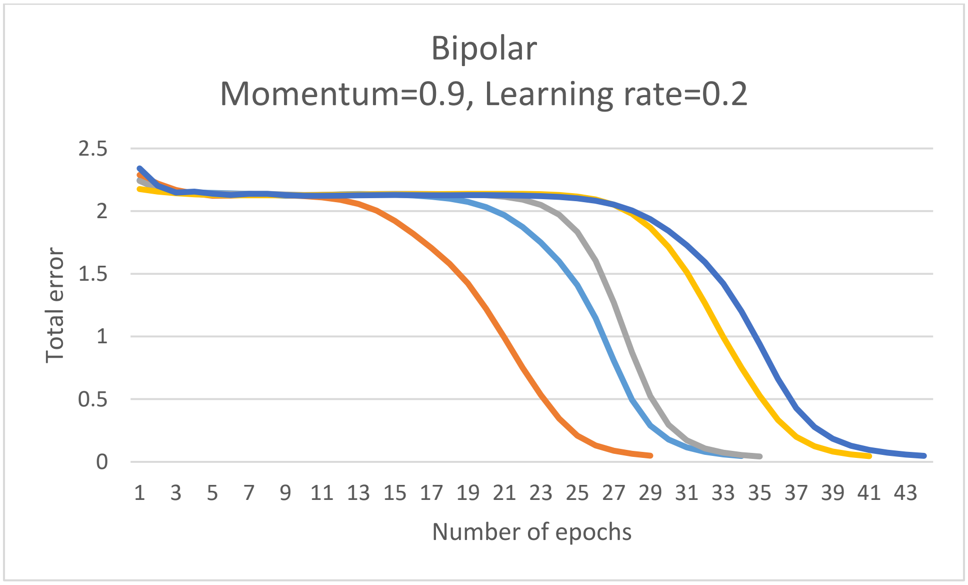
The average number to converge: 450

The maximum number to converge: 1367

The minimum number to converge: 206

The convergence rate: 100%

c.2) The graph below shows the conditions of five trials, they use a bipolar representation with momentum set to be 0.9 and learning rate set to be 0.2.



Graph c2. Bipolar representation with momentum 0.9 and learning rate 0.2

By implementing 10,000 times of trials, the statistics are achieved as below:

The average number to converge: 36

The maximum number to converge: 111

The minimum number to converge: 16

The convergence rate: 100%

Additional Statements

For bipolar, each time to update weights, if following the order of the lower layer to update weight, the rate of convergence will be around 40%. For example, when updating weights from input layer to hidden layer, if updating all weights connected to some specific input node and then updating weights connected to other input nodes, non-convergence will happen. But when updating all weights connected to some specific hidden node and then updating weights connected to other hidden nodes, non-convergence could be avoided and the rate could reach 100%.

Test:

For 10,000 trials with Bipolar representation, the momentum is set 0.0, the learning rate is set 0.2,

The average number to converge: 342 (count only if it is converge)

The maximum number to converge: 1069

The minimum number to converge: 162

The convergence rate: 35.31%

For 10,000 trials with Bipolar representation, the momentum is set 0.9, the learning rate is set 0.2,

The average number to converge: 38 (count only if it is converge)

The maximum number to converge: 158

The minimum number to converge: 19

The convergence rate: 46.34%

Appendix

1. BpNetWork class which implements the given interface

```java
package project_1_a;



import java.io.*;

public class BpNetWork implements NeuralNetInterface {

	int argNumInputs;
	int argNumHidden;
	double argLearningRate;
	double argMomentumTerm;
	double argA;
	double argB ;
	public double[][] node;
    public double[][] layError;
    public double[][][] nodeWeight;
    public double[][][] nodeWDelta;

	//double inputValue[]=new double[argNumInputs+1];
	BpNetWork(){
		argNumInputs=2;
		argNumHidden=4;
		argLearningRate=0.2;
		argMomentumTerm=0.0;
		argA=0;
		argB=1;
		/*for(int i=0;i<inputValue.length-1;i++){
			inputValue[i]=0;
		}
		inputValue[inputValue.length-1]=1;*/
	}
	BpNetWork(int[] nodenum, double rate, double mobp)
	{
		argNumInputs=nodenum[0];
		argNumHidden=nodenum[1];
		argA=0;
		argB=1;
		    this.argMomentumTerm = mobp;
		  this.argLearningRate = rate;
		  node = new double[nodenum.length][];
		  layError = new double[nodenum.length][];
		  nodeWeight = new double[nodenum.length][][];
		  nodeWDelta = new double[nodenum.length][][];
	}

	 int NumInputWBias=argNumInputs+1;
	 int NumHidWBias=argNumHidden+1;


	 /**
		* Return a bipolar sigmoid of the input X
		* @param x The input
		* @return f(x) = 2 / (1+e(-x)) - 1
		*/
	 public double sigmoid(double x){
		return 2/(1+Math.pow(Math.E, -x))-1;
	 };


	 /**
	 * This method implements a general sigmoid with asymptotes bounded by (a,b)
	 * @param x The input
	 * @return f(x) = b_minus_a / (1 + e(-x)) - minus_a
	 */
	 public double customSigmoid(double x){
		return (argB-argA)/(1+Math.pow(Math.E, -x))+argA;
	 };

	 /**
	 * Initialize the weights to random values.
	 * For say 2 inputs, the input vector is [0] & [1]. We add [2] for the bias.
	 * Like wise for hidden units. For say 2 hidden units which are stored in an
array.
	 * [0] & [1] are the hidden & [2] the bias.
	 * We also initialise the last weight change arrays. This is to implement
the alpha term.
	 */
	 public void initializeWeights(){
		int[] nodeNum=new int[3];
		nodeNum[0]=this.argNumInputs;
		nodeNum[1]=this.argNumHidden;
		nodeNum[2]=1;
		for(int l=0;l<nodeNum.length;l++){
		     node[l]=new double[nodeNum[l]];
		     layError[l]=new double[nodeNum[l]];
		     if(l+1<nodeNum.length){
			 nodeWeight[l]=new double[nodeNum[l]+1][nodeNum[l+1]];
```

```
                    nodeWDelta[l]=new double[nodeNum[l]+1][nodeNum[l+1]];
                    for(int j=0;j<nodeNum[l]+1;j++)
                        for(int i=0;i<nodeNum[l+1];i++)
                            nodeWeight[l][j][i]=Math.random()-0.5;
                }
            }
        }


        /**
        * Initialize the weights to 0.
        */

        public void zeroWeights(){
       int[] nodeNum=new int[3];
        nodeNum[0]=this.argNumInputs;
        nodeNum[1]=this.argNumHidden;
        nodeNum[2]=1;

        for(int l=0;l<nodeNum.length;l++){
             node[l]=new double[nodeNum[l]];
             layError[l]=new double[nodeNum[l]];
             if(l+1<nodeNum.length){
                 nodeWeight[l]=new double[nodeNum[l]+1][nodeNum[l+1]];
                 nodeWDelta[l]=new double[nodeNum[l]+1][nodeNum[l+1]];
                 for(int j=0;j<nodeNum[l]+1;j++)
                     for(int i=0;i<nodeNum[l+1];i++)
                         nodeWeight[l][j][i]=0;
             }
         }
    }




        ////commone interface part
        /**
         * @param X The input vector. An array of doubles.
         * @return The value returned by th LUT or NN for this input vector
         */
         public double outputFor(double [] X){
                for(int l=1;l<node.length;l++){
                   for(int j=0;j<node[l].length;j++){
                       double z=nodeWeight[l-1][node[l-1].length][j];
                       for(int i=0;i<node[l-1].length;i++){
                           node[l-1][i]=l==1?X[i]:node[l-1][i];
                           z+=nodeWeight[l-1][i][j]*node[l-1][i];
                       }
                       node[l][j]=this.customSigmoid(z);
                   }
               }

               return node[node.length-1][0];
         }



         /**
         * This method will tell the NN or the LUT the output
         * value that should be mapped to the given input vector. I.e.
         * the desired correct output value for an input.
         * @param X The input vector
        * @param argValue The new value to learn
        * @return The error in the output for that input vector
        */
        public double train(double [] X, double argValue){
        double out=outputFor(X);
        double[] val=new double[1];
        val[0]=argValue;

         int l=node.length-1;
        // System.out.print(l);
       for(int j=0;j<layError[2].length;j++)
           layError[2][j]=(val[j]-out)*1/(this.argB-this.argA)*(out-argA)*(argB-
out);

      /* l--;

       for(;l>=0;l--){
           for(int j=0;j<layError[l].length;j++){
               double z = 0.0;
               for(int i=0;i<layError[l+1].length;i++){
                   z=z+l>0?layError[l+1][i]*nodeWeight[l][j][i]:0;
                   nodeWDelta[l][j][i]=
this.argMomentumTerm*nodeWDelta[l][j][i]+this.argLearningRate*layError[l+1][i]*nod
e[l][j];
                   nodeWeight[l][j][i]+=nodeWDelta[l][j][i];
                   if(j==layError[l].length-1){
                       nodeWDelta[l][j+1][i]=
this.argMomentumTerm*nodeWDelta[l][j+1][i]+this.argLearningRate*layError[l+1][i];
                       nodeWeight[l][j+1][i]+=nodeWDelta[l][j+1][i];
                   }
               }
               layError[l][j]=z*1/(this.argB-this.argA)*(node[l][j]-argA)*(argB-
node[l][j]);
           }

       }*/

       //hid-output
       for(int j=0;j<this.argNumHidden;j++){
         double preW=nodeWeight[1][j][0];

nodeWeight[1][j][0]+=this.argMomentumTerm*nodeWDelta[1][j][0]+this.argLearningRate
*layError[2][0]*node[1][j];
         nodeWDelta[1][j][0]=nodeWeight[1][j][0]-preW;
       }

nodeWDelta[1][this.argNumHidden][0]=this.argMomentumTerm*nodeWDelta[1][this.argNum
Hidden][0]+this.argLearningRate*layError[2][0];
         nodeWeight[1][this.argNumHidden][0]+=nodeWDelta[1][this.argNumHidden][0];


       //input-hidden
       for(int j=0;j<this.argNumHidden;j++){
         layError[1][j]=layError[2][0]*1/(this.argB-this.argA)*(node[1][j]-
argA)*(argB-node[1][j])*nodeWeight[1][j][0];
```

```
        for(int i=0;i<this.argNumInputs;i++){

nodeWDelta[0][i][j]=this.argMomentumTerm*nodeWDelta[0][i][j]+this.argLearningRate*
layError[1][j]*node[0][i];
              nodeWeight[0][i][j]+=nodeWDelta[0][i][j];
        }

nodeWDelta[0][this.argNumInputs][j]=this.argMomentumTerm*nodeWDelta[0][this.argNum
Inputs][j]+this.argLearningRate*layError[1][j];

nodeWeight[0][this.argNumInputs][j]+=nodeWDelta[0][this.argNumInputs][j];

      }

      //System.out.print(node_weight);
       return val[0]-out;
}


        /**
         * A method to write either a LUT or weights of an neural net to a file.
         * @param argFile of type File.
        * @throws FileNotFoundException
        * @throws IOException
         */
        public void save(File argFile) throws IOException {

              try{
               FileOutputStream fp=new FileOutputStream(argFile);
               ObjectOutputStream op=new ObjectOutputStream(fp);
               op.writeObject(nodeWeight);
               op.close();
              }
              catch(IOException exc){
                    System.out.println("error");
                         return;
              }

        }


        /**
         * Loads the LUT or neural net weights from file. The load must of course
         * have knowledge of how the data was written out by the save method.
         * You should raise an error in the case that an attempt is being
         * made to load data into an LUT or neural net whose structure does not
match
         * the data in the file. (e.g. wrong number of hidden neurons).
         * @throws IOException
         */

        public void load(String argFileName) throws IOException{
             FileInputStream fin;
             try{
                   fin = new FileInputStream(argFileName);
             }catch(FileNotFoundException exc){
                   System.out.println("file not found");
                   return;
             }
```

```
            int[] nodenum=new int[3];
            nodenum[0]=this.argNumInputs;
            nodenum[1]=this.argNumHidden;
            nodenum[2]=1;

            for(int l=0;l<nodenum.length;l++){
                 node[l]=new double[nodenum[l]];
                 layError[l]=new double[nodenum[l]];
                 if(l+1<nodenum.length){
                     nodeWeight[l]=new double[nodenum[l]+1][nodenum[l+1]];
                     nodeWDelta[l]=new double[nodenum[l]+1][nodenum[l+1]];
                     for(int j=0;j<nodenum[l]+1;j++)
                         for(int i=0;i<nodenum[l+1];i++)
                             nodeWeight[l][j][i]=fin.read();
                 }
             }
             fin.close();
        }
}
```

2. Test class with the main function and including the binary part and the bipolar part.

```
package project_1_a;


import java.io.File;

import java.io.IOException;


public class NeuralNetTest {


 public static void main(String[] args) throws IOException{

        int[] layernum=new int[3];

        layernum[0]=2;

        layernum[1]=4;

        layernum[2]=1;

        double mobp=0.9;

        double rate=0.2;


        double mobp1=0.0;

        double rate1=0.2;
        final int epoch=10000;
        double[][] in1={{0,0},{0,1},{1,0},{1,1}};
        double[] out1={0,1,1,0};
        double[] X1=new double[2];

        double[][] in2={{-1,-1},{-1,1},{1,-1},{1,1}};
        double[] out2={-1,1,1,-1};
        double[] X2=new double[2];
        int countNum=10;
        int sumBinary=0;
        int aveBinary=0;
        int maxBinary=0;
        int minBinary=epoch;
         for(int c=0;c<countNum;c++){
        int i=0;
       double[] error1=new double[epoch];
                    BpNetWork myNet=new BpNetWork(layernum,rate,mobp);
                    myNet.initializeWeights();
        for(;i<epoch;i++){
                   for(int j=0;j<4;j++){
          X1[0]=in1[j][0];
                   X1[1]=in1[j][1];
                   double argValue=out1[j];
                   double trainOut=myNet.train(X1,argValue);
                   error1[i]+=Math.pow(trainOut, 2)/2;
                   }
                   //System.out.print(error1[i]);
                   //System.out.print("\t");
                   //System.out.print(error1[i]);
                   if(error1[i]<0.05)
                           break;
```

```
              }
              if(i>maxBinary) maxBinary=i;
                     if(i<minBinary) minBinary=i;
              sumBinary+=i;
              //System.out.print("\n");
              //System.out.print(i);
              }
               aveBinary=sumBinary/countNum;
               System.out.print("\n");
              System.out.println("average number:"+aveBinary+"\tmax number:"+maxBinary+"\tmin
number:"+minBinary);

             //System.out.println("\n");

             int countConv=0;
              int maxBipolar=0;
              int minBipolar=epoch;
              int sumBipolar=0;
              int aveBipolar=0;
              for(int c=0;c<countNum;c++){

                      BpNetWork myNet1=new BpNetWork(layernum,rate1,mobp1);
                      myNet1.initializeWeights();
                      myNet1.argA=-1;
                      int m=0;
                      double[] error2=new double[epoch];
              for(;m<epoch;m++){
                     for(int n=0;n<4;n++){
                X2[0]=in2[n][0];
                     X2[1]=in2[n][1];
                     double argValue2=out2[n];
                     double trainOut2=myNet1.train(X2,argValue2);
                     error2[m]+=Math.pow(trainOut2, 2)/2;
                     }
                     //System.out.print(error2[m]);
                     //System.out.print(error2[m]);
                     //System.out.print("\t");

                     if(error2[m]<0.05)
                             {
                             countConv++;
                             break;

                             }

              }
             // if(m<epoch)
                     // countConv++;
              //System.out.print("\n");
              if(m!=10000)
              sumBipolar+=m;
              if(m!=10000&&m>maxBipolar) maxBipolar=m;
                     if(m<minBipolar) minBipolar=m;
              //System.out.print(m);
              //System.out.print("\n");
              }
              aveBipolar=sumBipolar/countNum;
        System.out.print("average number:"+aveBipolar+"\tmax number:"+maxBipolar+"\tmin
number:"+minBipolar+"\tconverge time:"+countConv);
```

```
        myNet.save(new File("C:/Users/wangzhen/Desktop/EECE592/project_1_a/bin/weight.txt"));

    }
    }
```