Lab Sheet 5
Write a program to demonstrate the back propagation algorithm.

Backpropagation.java

```java
public class Backpropagation {
    public final static int numInputs = 4;
    public final static int numPatterns = 4;
    public final static int numHidden = 5;
    public final static int numEpochs = 300;
    public final static double LR_IH = 0.7;
    public final static double LR_HO = 0.07;
    private int patNum = 0;
    private double errThisPat = 0.0;
    private double outPred = 0.0;
    private double RMSerror = 0.0;
    // the outputs of the hidden neurons
    private double[] hiddenVal= new double[numHidden];
    // the weights
    private double[][] weightsIH = new double[numInputs][numHidden];
    private double[] weightsHO = new double[numHidden];
    // the data
    private int[][] trainInputs = new int[numPatterns][numInputs];
    private int[] trainOutput = new int[numPatterns];
    public void initWeights(){
        for(int j = 0;j<numHidden;j++){
            weightsHO[j] = (getRand() - 0.5)/2;
            for(int i = 0;i<numInputs;i++){
                weightsIH[i][j] = (getRand() - 0.5)/5;
                System.out.printf("Weight = %f\n", weightsIH[i][j]);
            }
        }
    }
    public double getRand(){
        return Math.random();
    }
    public void initData(){
        System.out.println("Initialising Data");
        // the data here is the XOR data
        // it has been rescaled to the range
        // [-1][1]
        // an extra input valued 1 is also added
        // to act as the bias
        // the output must lie in the range -1 to 1
        trainInputs[0][0] = 1;
        trainInputs[0][1] = -1;
        trainInputs[0][2] = 1; //bias
        trainOutput[0] = 1;
        trainInputs[1][0] = -1;
        trainInputs[1][1] = 1;
        trainInputs[1][2] = 1; //bias
        trainOutput[1] = 1;
        trainInputs[2][0] = 1;
        trainInputs[2][1] = 1;
        trainInputs[2][2] = 1; //bias
        trainOutput[2] = -1;
        trainInputs[3][0] = -1;
        trainInputs[3][1] = -1;
        trainInputs[3][2] = 1; //bias
        trainOutput[3] = -1;
```

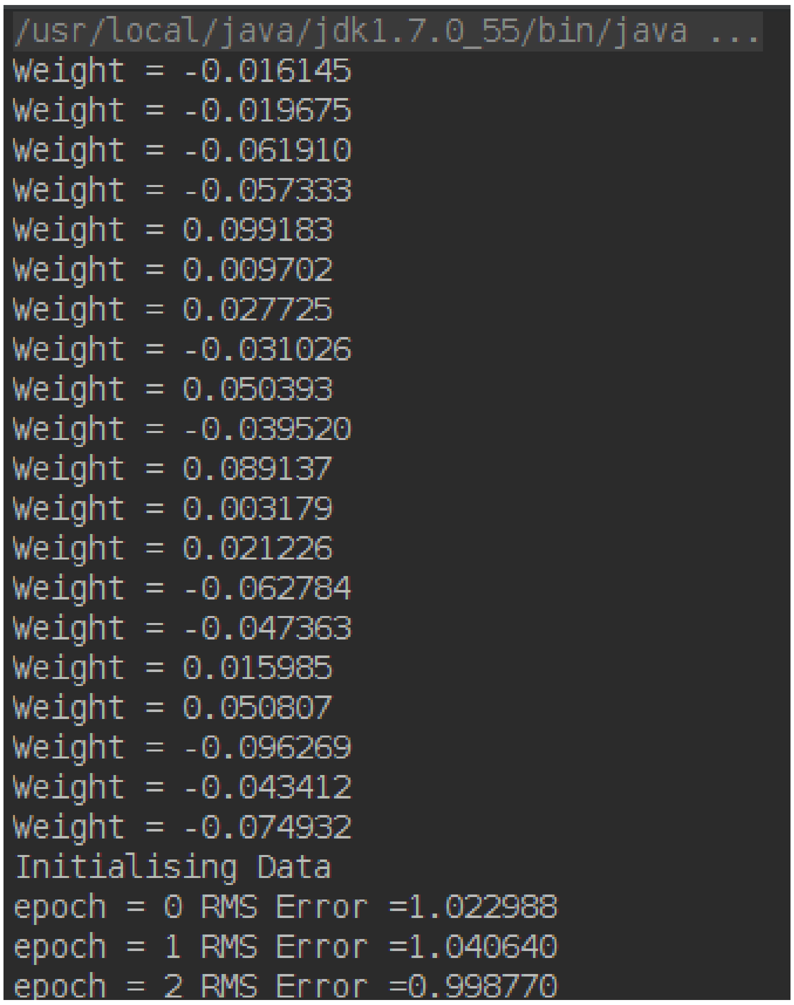
```java
}
public void train(){
    for(int j = 0;j <= numEpochs;j++){
        for(int i = 0;i<numPatterns;i++){
            //select a pattern at random
            patNum = (int)(Math.random()*numPatterns);
            //calculate the current network output
            //and error for this pattern
            calcNet();
            //change network weights
            WeightChangesHO();
            WeightChangesIH();
        }
        //display the overall network error
        //after each epoch
        calcOverallError();
        System.out.printf("epoch = %d RMS Error =%f\n",j,RMSerror);
    }
}
// calculates the network output
public void calcNet(){
    //calculate the outputs of the hidden neurons
    //the hidden neurons are tanh
    int i = 0;
    for(i = 0;i<numHidden;i++){
        hiddenVal[i] = 0.0;
        for(int j = 0;j<numInputs;j++){
            hiddenVal[i] = hiddenVal[i] + (trainInputs[patNum][j] *
                    weightsIH[j][i]);
        }
        hiddenVal[i] = Math.tanh(hiddenVal[i]);
    }
    //calculate the output of the network
    //the output neuron is linear
    outPred = 0.0;
    for(i = 0;i<numHidden;i++){
        outPred = outPred + hiddenVal[i] * weightsHO[i];
    }
    //calculate the error
    errThisPat = outPred - trainOutput[patNum];
}
//adjust the weights hidden-output
public void WeightChangesHO(){
    for(int k = 0;k<numHidden;k++){
        double weightChange = LR_HO * errThisPat * hiddenVal[k];
        weightsHO[k] = weightsHO[k] - weightChange;
        //regularisation on the output weights
        if (weightsHO[k] < -5){
            weightsHO[k] = -5;
        }
        else if (weightsHO[k] > 5){
            weightsHO[k] = 5;
        }
    }
}
// adjust the weights input-hidden
public void WeightChangesIH(){
    for(int i = 0;i<numHidden;i++){
        for(int k = 0;k<numInputs;k++){
            double x = 1 - (hiddenVal[i] * hiddenVal[i]);
            x = x * weightsHO[i] * errThisPat * LR_IH;
            x = x * trainInputs[patNum][k];
```

```
                double weightChange = x;
                weightsIH[k][i] = weightsIH[k][i] - weightChange;
            }
        }
    }
    // calculate the overall error
    public void calcOverallError(){
        RMSerror = 0.0;
        for(int i = 0;i<numPatterns;i++){
            patNum = i;
            calcNet();
            RMSerror = RMSerror + (errThisPat * errThisPat);
        }
        RMSerror = RMSerror/numPatterns;
        RMSerror = Math.sqrt(RMSerror);
    }
    // display results
    public void displayResults(){
        for(int i = 0;i<numPatterns;i++){
            patNum = i;
            calcNet();
            System.out.printf("pat = %d actual = %d neural model = 
%f\n",patNum+1,trainOutput[patNum],outPred);
        }
    }
}
```

BackpropagationMain.java

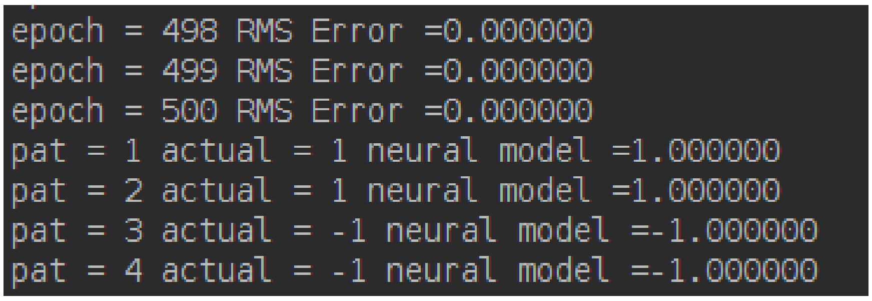
```
public class BackMain {
    public static void main(String[] args) {
        Backpropagation backpropagation = new Backpropagation();
        // initiate the weights
        backpropagation.initWeights();
        // load in the data
        backpropagation.initData();
        // train the network
        backpropagation.train();
        //training has finished
        //display the results
        backpropagation.displayResults();
    }
}
```

Output:



```
/usr/local/java/jdk1.7.0_55/bin/java ...
Weight = -0.016145
Weight = -0.019675
Weight = -0.061910
Weight = -0.057333
Weight = 0.099183
Weight = 0.009702
Weight = 0.027725
Weight = -0.031026
Weight = 0.050393
Weight = -0.039520
Weight = 0.089137
Weight = 0.003179
Weight = 0.021226
Weight = -0.062784
Weight = -0.047363
Weight = 0.015985
Weight = 0.050807
Weight = -0.096269
Weight = -0.043412
Weight = -0.074932
Initialising Data
epoch = 0 RMS Error =1.022988
epoch = 1 RMS Error =1.040640
epoch = 2 RMS Error =0.998770
```

.
.
.
.
.



```
epoch = 498 RMS Error =0.000000
epoch = 499 RMS Error =0.000000
epoch = 500 RMS Error =0.000000
pat = 1 actual = 1 neural model =1.000000
pat = 2 actual = 1 neural model =1.000000
pat = 3 actual = -1 neural model =-1.000000
pat = 4 actual = -1 neural model =-1.000000
```