Submission Information

Course:	CS 7375 – Artificial Intelligence
Student Name:	Marion Garrett Sisk
Student ID:	000942002
Assignment #:	3
Due Date:	05/01/21
Signature:	
Score:	

Table of Contents

Agent Design

The artificial neural network created as part of this assignment has a 5-1-2 arrangement. With four input neurons (including 1 bias neuron), one hidden neuron, and two output neurons. The simplicity of this arrangement causes the complex linear algebra methods to simplify down to a simple dot product of two vectors, because of this, no linear algebra libraries were used, as it would be vastly too cumbersome to implement those, versus merely hand coding a dot product.

The agent is designed to work through the use of a console interface. In this interface you can change the learning rate, and training epoch limits. These parameters control how fast and how much learning is done during the backpropagation phase of the ANN's operation. The test set is hard coded, and the individual pixels are represented by binary digits in an array. A "1" indicates the pixel is "dark," a "0" indicates the pixel is "light." The first element of the test array

is the bias neuron, the next four are the actual pixel inputs. The test set has two additional binary values in an array, these are the correct answers. The sixth element in the array denotes the four-pixel structure is “dark” if it contains anything other than a zero. Likewise, the seventh element denotes the structure is “light” if it contains anything other than a zero.

The test set is simply the training set, minus the two last digits denoting the actual answer. The backpropagation algorithm uses the root mean square error when computing error values, and then propagates them back through the ANN to the inputs using the expressions detailed in the lecture slides.

Output Screenshot

Main Menu upon startup:

```
*************** Simple Artificial Neural Network ***************
*                     Marion Garrett Sisk                      *
*                    CS 7375 Assignment 03                     *
****************************************************************

Current Parameters:
Learning Rate: 0.1
Training Epochs: 50000

-------- Main Menu --------
    1: Change Learning Rate
    2: Change total training epochs
    3: Train Artificial Neural Network
    4: Test Artificial Neural Network
    0: Exit Program

Enter your choice:
```

Pre-Training Results Example:

```
**********
Test scenario 12
Manual Determination = Light
*** ANN Determinations
Light: 0.5053397603463887
Dark: 0.5502643928194385
Image is Dark.

**********
Test scenario 13
Manual Determination = Dark
*** ANN Determinations
Light: 0.5049849059332675
Dark: 0.5469442017303028
Image is Dark.
```

Post-Training Results Example:

```
**********
Test scenario 12
Manual Determination = Light
*** ANN Determinations
Light: 0.9905766183846473
Dark: 0.00942338513114853
Image is Light.

**********
Test scenario 13
Manual Determination = Dark
*** ANN Determinations
Light: 0.5004777818994729
Dark: 0.4995222181391842
Image is Light.
```

Tasks the Agent Can Solve

This particular agent can solve for the four-pixel input given in the actual assignment. It cannot solve for much else, since the ANN was trained for this specific purpose. Even then, when it determines the cell isn't light, it is still straddling the 50/50 mark, with a slight edge towards light, when it should be dark. The opposite, however, is not true. The ANN predicts when the four-pixel array is "light" with almost a total 100/0 output array.

Video Demonstration

A video demonstration of the application can be found at the following YouTube link:

https://youtu.be/B2xDiC4z6OA

GitHub Repository

All source files and associated binary files can be found on my personal GitHub page at:

https://github.com/mgarrettsisk/neuralNetwork

Source Code

assignment03.java

```
package mainApplication;

import java.util.Scanner;

public class assignment03 {

    public static void main(String[] args) {
        // this is the entry point to the assignment03 application to train
        // sentinel loop that allows for user to run program again, or end process

        // Create variables for the program to function correctly
        neuralNetwork ann = new neuralNetwork();
        double learnRate = 0.1;
        int epochs = 50000;
        double[][] trainSet = {
                {1,0,0,0,0,0,1},
                {1,0,0,0,1,0,1},
                {1,0,0,1,0,0,1},
                {1,0,0,1,1,0,1},
                {1,0,1,0,0,0,1},
                {1,0,1,0,1,0,1},
                {1,0,1,1,0,0,1},
                {1,0,1,1,1,1,0},
                {1,1,0,0,0,0,1},
                {1,1,0,0,1,0,1},
                {1,1,0,1,0,0,1},
                {1,1,0,1,1,1,0},
                {1,1,1,0,0,0,1},
                {1,1,1,0,1,1,0},
                {1,1,1,1,0,1,0},
                {1,1,1,1,1,1,0}
        };

        double[][] testSet = {
                {1,0,0,0,0},
                {1,0,0,0,1},
                {1,0,0,1,0},
                {1,0,0,1,1},
                {1,0,1,0,0},
                {1,0,1,0,1},
                {1,0,1,1,0},
                {1,0,1,1,1},
                {1,1,0,0,0},
                {1,1,0,0,1},
                {1,1,0,1,0},
                {1,1,0,1,1},
                {1,1,1,0,0},
                {1,1,1,0,1},
                {1,1,1,1,0},
                {1,1,1,1,1}
        };
```

```
        Scanner programControl = new Scanner(System.in);
        int runProgram = -1;
        System.out.println("**************** Simple Artificial Neural Network ***************");
        System.out.println("*                      Marion Garrett Sisk                      *");
        System.out.println("*                    CS 7375 Assignment 03                      *");
        System.out.println("*****************************************************************");

        while (runProgram != 0) // whole program runs in a while loop dependent on value of runProgram
        {
            // Main Menu Dialog
            System.out.println();
            System.out.println("Current Parameters: ");
            System.out.println("Learning Rate: " + learnRate);
            System.out.println("Training Epochs: " + epochs);
            System.out.println();
            System.out.println("-------- Main Menu --------");
            System.out.println("\t1: Change Learning Rate");
            System.out.println("\t2: Change total training epochs");
            System.out.println("\t3: Train Artificial Neural Network");
            System.out.println("\t4: Test Artificial Neural Network");
            System.out.println("\t0: Exit Program");
            System.out.println();
            System.out.print("Enter your choice: "); // prompt user for their choice
            runProgram = programControl.nextInt(); // overwrite runProgram string w/ entered value
            System.out.println();

            // Program modes, based upon which
            switch(runProgram) {
                case(1):
                    Scanner learning = new Scanner(System.in);
                    System.out.print("Enter a new learning rate between 0 and 1: ");
                    learnRate = learning.nextDouble();
                    System.out.println("The learning rate is now " + learnRate);
                    System.out.println();
                    break;
                case(2):
                    Scanner epochNum = new Scanner(System.in);
                    System.out.print("Enter a new training epoch limit (integer values): ");
                    epochs = epochNum.nextInt();
                    System.out.println("The training epoch limit is now " + epochs);
                    System.out.println();
                    break;
                case(3):
                    System.out.println("Training the neural network using the training set.");
                    ann.train(trainSet,epochs,learnRate);
                    break;
                case(4):
                    System.out.println("Iterating through all test scenarios.");
                    for (int i = 0; i < testSet.length; i++) {
                        System.out.println("********************************");
                        System.out.println("Test scenario " + i);
                        //System.out.println(testSet[i].toString());
                        if (isDark(testSet[i])) {
                            System.out.println("Manual Determination = Dark");
                        } else {
                            System.out.println("Manual Determination = Light");
                        }
                        System.out.println("*** ANN Determinations");
                        displayType(ann.feedForward(testSet[i]));
                        System.out.println();
                        System.out.println();
                    }
                    break;
            }
            System.out.println();
        }
    }

    public static void displayType(double[] results) {
        // this method takes the results from a feed forward operation and prints a result to the
console
        System.out.println("Light: " + results[1]);
        System.out.println("Dark: " + results[0]);
        if (results[0] > results[1]) {
            System.out.print("Image is Dark.");
        } else
            System.out.print("Image is Light.");
```

```
    }

    public static boolean isDark(double[] testInput) {
        // this method takes a sum of entries of the array to manually check whether the image should
be light
        // or dark. This is used to compare the determination done by the ANN
        double arraySum = 0;
        for (int j = 0; j < testInput.length; j++) {
            arraySum = arraySum + testInput[j];
        }
        if (arraySum > 3) {
            return true;
        } else {
            return false;
        }
    }
}
```

neuralNetwork.java

```
package mainApplication;

import java.util.Random;

public class neuralNetwork {
    // This class builds an artificial neural network with 4 input neurons (plus 1 bias neuron), 1 hidden neuron, and 2
    // output neurons. The test implementation allows for the determination of a 2x2 matrix with either 0 or 1 in each
    // cell. A 1 would correspond to a "dark" pixel, and a 0 would correspond to a "light" pixel. The ANN should be
able
    // to determine whether a given combination of the 2x2 matrix is "light" or "dark" based on a training set.

    // attributes
    private double[] inputArray = {1,0,0,0,0};
    private double[] w1 = {0.0, 0., 0.5, 0.5, 0.5};
    private double a1 = 0.0;
    private double[] w2 = {0.5, 0.5};
    private double[] a2 = {0,0};

    neuralNetwork() {
        generateRandomWeights();
    }

    // public methods
    public double[] feedForward(double[] inputVector) {
        // For input layer to hidden layer, we multiply the input vector by the weight matrix and sum the values.
        //System.out.println("Feed Forward Operation begun...");
        double inputSum = 0.0;
        for (int i = 0; i < inputVector.length; i++) {
           inputSum = inputSum + inputVector[i]*this.w1[i];
        }
        // activation of hidden neuron from 5 input neurons
        a1 = sigmoid(inputSum);
        //System.out.println(" h_1 = " + a1);

        // From hidden layer (1 neuron) to output layer (2 neurons)
        a2[0] = sigmoid(a1*w2[0]);
        a2[1] = sigmoid(a1*w2[1]);
        //System.out.println("a2[0] = " + a2[0]);
        //System.out.println("a2[1] = " + a2[1]);

        // returns the output
        return a2;
    }

    public void train(double[][] trainingSet, int epochs, double learningRate) {
        // This method takes a training set of values and iterates over them using the backpropagation algorithm to set
        // the appropriate weights. It will iterate over the training set for the provided number of epochs.
        //
        //  The training set should have the following values:
        //
        //      T = {{bias, x0y0, x1y0, x0y1, x1y1, dark, light}, ... }
        //
        //  In this instance, the bias is always equal to 1. The dark or light is either a 1 or a 0 depending upon the
        //  configuration of the pixels themselves.

        int iteration = 0;

        while (iteration < epochs) {
            if (iteration%100 == 0) {
                System.out.println("Entering training epoch " + iteration);
            }
            // for each element in the training set
            for (int i = 0; i < trainingSet.length; i++) {

                // run the test vector through the feedForward method and obtain results
```

```
                double[] feedFwdTrainingSet = {
                        trainingSet[i][0],
                        trainingSet[i][1],
                        trainingSet[i][2],
                        trainingSet[i][3],
                        trainingSet[i][4]
                };
                //System.out.println("---- Training Feed Forward Call ----");
                double[] results = feedForward(feedFwdTrainingSet);
                //System.out.println("---- Training Feed Forward Call Finished ----");
                // calculate error for each output
                double[] outputErrors = {
                        outputError(trainingSet[i][5], results[0]),
                        outputError(trainingSet[i][6], results[1])
                };
                //System.out.println("Output Errors: \tO_1 = " + outputErrors[0] + ", O_2 = " + outputErrors[1]);

                // back propagate the errors to the hidden neuron unit
                // compute the weighted sum of errors from outputs
                double errorSum = outputErrors[0]*this.w2[0] + outputErrors[1]*this.w2[1];
                // compute the weighted error for back propagation to input neurons
                double inputError = this.a1*(1-this.a1)*errorSum;

                // compute the change in weights between inputs and hidden layer and adjust weights accordingly
                double[] inputDeltas = {
                        learningRate*inputError*trainingSet[i][0],
                        learningRate*inputError*trainingSet[i][1],
                        learningRate*inputError*trainingSet[i][2],
                        learningRate*inputError*trainingSet[i][3],
                        learningRate*inputError*trainingSet[i][4]
                };
                // alter weights in w1 with deltas from above
                for (int j = 0; j < w1.length; j++) {
                    this.w1[j] = this.w1[j] + inputDeltas[j];
                }

                // compute the change in weights between outputs and hidden layer and adjust weights accordingly
                double[] outputDeltas = {
                        learningRate*outputErrors[0]*this.a1,
                        learningRate*outputErrors[1]*this.a1
                };
                // alter weights in w2 with deltas from above
                for (int k = 0; k < w2.length; k++) {
                    this.w2[k] = this.w2[k] + outputDeltas[k];
                }
                // then repeat the process for the next set of test values
            }
            iteration++;
        }
    }

    // private methods
    private double sigmoid(double input) {
        // this method takes a double value as input and returns the equivalent sigmoid value as another double
        return (1/(1+Math.exp(-input)));
    }

    private double outputError(double expected, double actual) {
        // this method takes the expected value and the actual value and calculates the error based on the
        return actual*(1-actual)*(expected-actual);
    }

    private void generateRandomWeights() {
        // this method generates random starting weights for arrays w1 and w2 in this object. This is only called in
the
        // constructor method upon instantiation
        Random rand = new Random();
        for (int i = 0; i < this.w1.length; i++) {
            this.w1[i] = (rand.nextDouble() - 0.5);
        }
        for (int i = 0; i < this.w2.length; i++) {
            this.w2[i] = (rand.nextDouble() - 0.5);
        }
    }
}
```