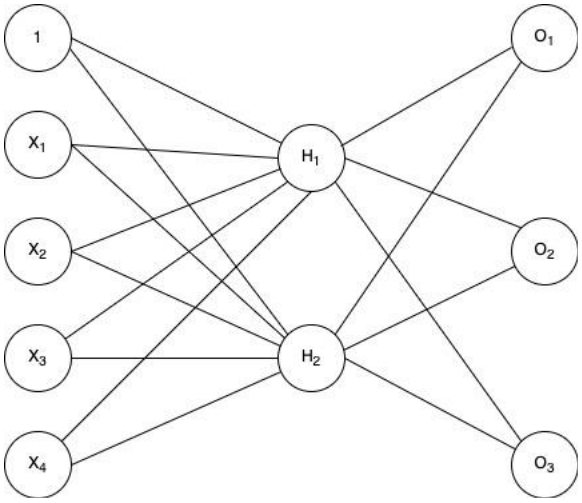
I: Your information.
// Course: CS3642_
// Student name: Andujar Brutus
// Student ID: 0001002015
// Assignment #: Test 2
// Due Date: 11/25/2023
// Signature: Andujar Brutus (Your signature assures that everything is your own work. Required)
// Score: _____ (Note: Score will be posted on D2L)

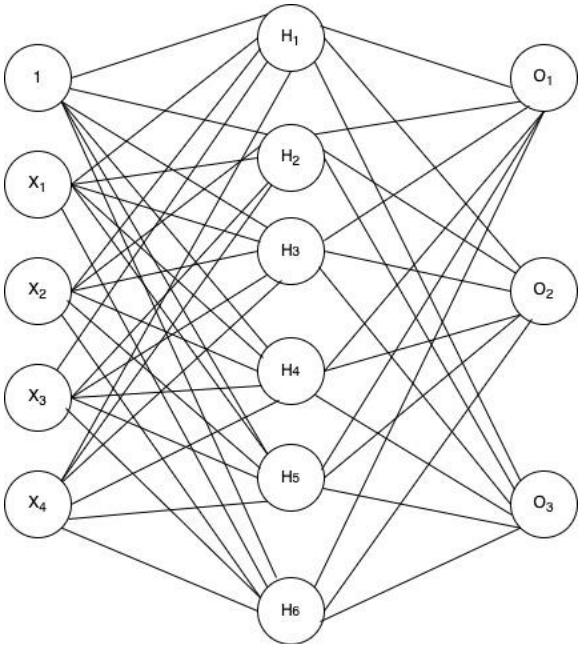
Topology	Accuracy	Time Complexity
5-2-3	89.33%	O(n)
5-6-3	89.33%	O(n)

Though the accuracies are equal for both the 5-2-3 ANN and the 5-6-3 ANN, the calculated network errors are different. The 5-2-3 ANN seems to have an average network error value of 22.3 while the 5-6-3 ANN has an average network error value of 19.8. This shows that the 5-6-3 ANN has a lower capacity for error though the accuracy was equivalent. It can be assumed the lack of change in accuracy is due to the overlapping nature of the iris data points.

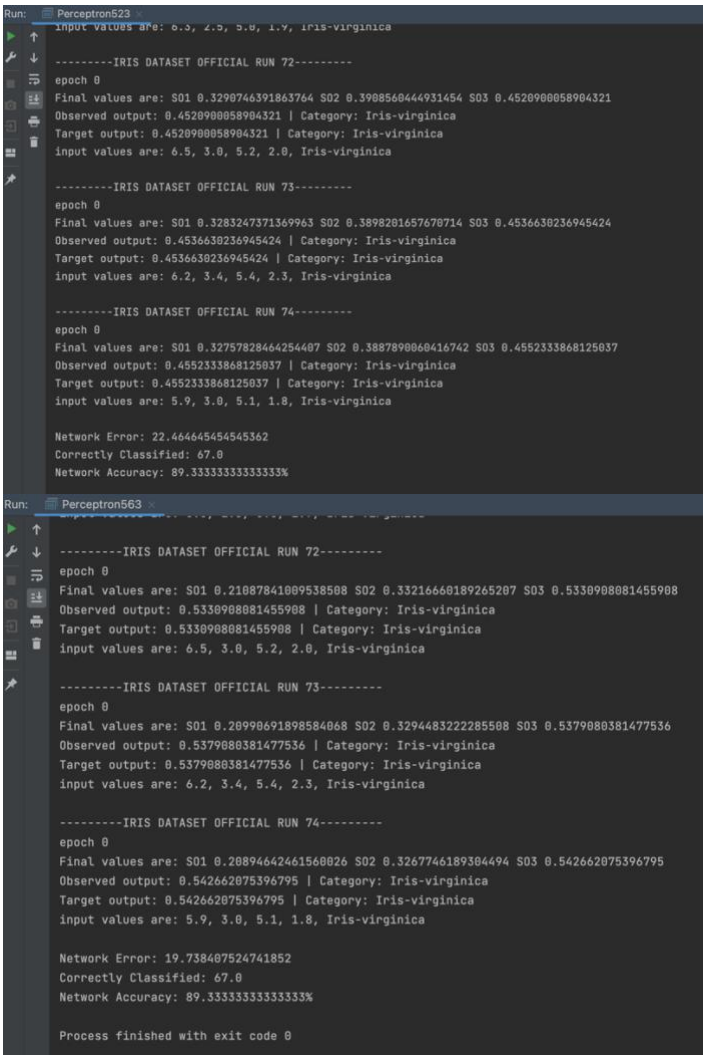
5-2-3 Diagram



5-6-3 Diagram



Screenshots:



GUI:

II: Your architecture. Please briefly explain.

5-2-3 ANN code:

This code was designed using the multi-layer neural network process. I found that the formulaic notations were varied and often used different symbols to describe the same calculation. Despite all the detailed analysis of the artificial neural network building process, I find the output to be straightforward and reliable. Attention to detail was required for both parts of the test 2 but especially part 1 in which we were required to build the initial ANN from scratch. I have learned a lot as I have had to program this over the course of a couple days. Attention to detail is crucial during this process.

```
package com.example.aitest2ui;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.io.File;
import java.util.*;
import java.lang.*;

@CrossOrigin(origins = "http://localhost:3000/")
@RestController
@RequestMapping(value = "/ANN523")
public class Perceptron523 {
    Scanner testInput = new Scanner(System.in);

    //initialize values
    static List<String> irisLines = new ArrayList<>();
    static List<String[]> irisValues = new ArrayList<>();
    static String setosaName = "";
    static double setosaSepalLengthAvg = 0, setosaSepalWidthAvg = 0,
setosaPetalLengthAvg = 0, setosaPetalWidthAvg = 0, setosaTotalAvg = 0;
    static double setosaSepalLengthMin = 10, setosaSepalLengthMax = 0,
setosaSepalWidthMin = 10, setosaSepalWidthMax = 0, setosaPetalLengthMax = 0,
setosaPetalLengthMin = 10, setosaPetalWidthMax = 0, setosaPetalWidthMin = 10;
    static String versicolourName = "";
    static double versicolourSepalLengthAvg = 0, versicolourSepalWidthAvg =
0, versicolourPetalLengthAvg = 0, versicolourPetalWidthAvg = 0,
versicolourTotalAvg = 0;
    static double versicolourSepalLengthMin = 10, versicolourSepalLengthMax =
0, versicolourSepalWidthMin = 10, versicolourSepalWidthMax = 0,
versicolourPetalLengthMax = 0, versicolourPetalLengthMin = 10,
versicolourPetalWidthMax = 0, versicolourPetalWidthMin = 10;

    static String virginicaName = "";
    static double virginicaSepalLengthAvg = 0, virginicaSepalWidthAvg = 0,
virginicaPetalLengthAvg = 0, virginicaPetalWidthAvg = 0, virginicaTotalAvg =
0;
    static double virginicaSepalLengthMin = 10, virginicaSepalLengthMax = 0,
virginicaSepalWidthMin = 10, virginicaSepalWidthMax = 0,
virginicaPetalLengthMax = 0, virginicaPetalLengthMin = 10,
virginicaPetalWidthMax = 0, virginicaPetalWidthMin = 10;
    static double setosaPoints = 0;
    static double versicolourPoints = 0;
```

```
    static double virginicaPoints = 0;
    //array to store input positions
    private final double[] input = new double[4];
    private static double targetOutput, observedOutput;
    private final int BIAS_NEURON = 1;
    private double x1;
    private double x2;
    private double x3;
    private double x4;
    private String observedOutputName;
    private String targetOutputName;

    //Weights for BackPropagation
    //SOx being the output layer neuron and weight x being the hidden layer
neuron
    static double SO1weight1 = (Math.random() - .5);
    static double SO1weight2 = (Math.random() - .5);
    static double SO2weight1 = (Math.random() - .5);
    static double SO2weight2 = (Math.random() - .5);
    static double SO3weight1 = (Math.random() - .5);
    static double SO3weight2 = (Math.random() - .5);

    //SHx being the hidden layer neuron and weight x being the input layer
neuron
    static double SH1weight0 = (Math.random() - .5);
    static double SH1weight1 = (Math.random() - .5);
    static double SH1weight2 = (Math.random() - .5);
    static double SH1weight3 = (Math.random() - .5);
    static double SH1weight4 = (Math.random() - .5);
    static double SH2weight0 = (Math.random() - .5);
    static double SH2weight1 = (Math.random() - .5);
    static double SH2weight2 = (Math.random() - .5);
    static double SH2weight3 = (Math.random() - .5);
    static double SH2weight4 = (Math.random() - .5);

    static double summation1 = 0, summation2 = 0;
    double sigmaSO1, sigmaSO2, sigmaSO3, sigmaSH1, sigmaSH2;
    double setosaTargetOutput, versicolourTargetOutput,
virginicaTargetOutput;
    double O1OutputLayerError, O2OutputLayerError, O3OutputLayerError,
H1OutputLayerError, H2OutputLayerError;
    double networkError, OutputLayerNetError, correctlyClassified,
networkAccuracy;

    //default constructor
    Perceptron523() {
    }

    //get observed output name
    String getObservedOutputName() {
        return this.observedOutputName;
    }

    //set observed output name
    void setObservedOutputName(String newOutputName) {
        this.observedOutputName = newOutputName;
    }
```

```
    public double getX1() {
        return x1;
    }

    public void setX1(double x1) {
        this.x1 = x1;
    }

    public double getX2() {
        return x2;
    }

    public void setX2(double x2) {
        this.x2 = x2;
    }

    public double getX3() {
        return x3;
    }

    public void setX3(double x3) {
        this.x3 = x3;
    }

    public double getX4() {
        return x4;
    }

    public void setX4(double x4) {
        this.x4 = x4;
    }

    //initialize x values manually
    String inputPerceptron() {
        //receive X input values
        System.out.println("Sepal Length (cm):");
        setX1(testInput.nextDouble());
        System.out.println("Sepal Width (cm):");
        setX2(testInput.nextDouble());
        System.out.println("Petal Length (cm):");
        setX3(testInput.nextDouble());
        System.out.println("Petal Width (cm):");
        setX4(testInput.nextDouble());

        //accept values for each pixel
        setX1(x1);
        setX2(x2);
        setX3(x3);
        setX4(x4);

        //assign each value to an array position
        input[0] = getX1();
        input[1] = getX2();
        input[2] = getX3();
        input[3] = getX4();
```

```
        return "The input values are " + x1 + ", " + x2 + ", " + x3 + ", " +
x4 + ".";
    }

    //Hidden layer 1 calculations
    double SH1() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH1weight0) + (this.x1 * SH1weight1) +
(this.x2 * SH1weight2) + (this.x3 * SH1weight3) + (this.x4 * SH1weight4));
    }

    //Hidden layer 2 calculations
    double SH2() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH2weight0) + (this.x1 * SH2weight1) +
(this.x2 * SH2weight2) + (this.x3 * SH2weight3) + (this.x4 * SH2weight4));
    }

    //Sigma calculation to output from hidden layer
    double sigmaS(double SHx) {
        //used -1 rather than -SHx to avoid error
        return (1 / (1 + Math.exp(-1 * SHx)));
    }

    double sigmaSH1() {
        sigmaSH1 = sigmaS(SH1());
        return sigmaSH1;
    }

    double sigmaSH2() {
        sigmaSH2 = sigmaS(SH2());
        return sigmaSH2;
    }

    //Calculate weighted sums into the output layer (arbitrary weights)
    double SO1() {
        double SO1;
        SO1 = ((SO1weight1 * sigmaSH1()) + (SO1weight2 * sigmaSH2()));
        return SO1;
    }

    double SO2() {
        double SO2;
        SO2 = ((SO2weight1 * sigmaSH1()) + (SO2weight2 * sigmaSH2()));
        return SO2;
    }

    double SO3() {
        double SO3;
        SO3 = ((SO3weight1 * sigmaSH1()) + (SO3weight2 * sigmaSH2()));
        return SO3;
    }

    //Calculate the ANN (output from the output layer)
    double finalSigma(double SOx) {
        return sigmaS(SOx);
    }
```

```
    void setO1Output() {
        sigmaSO1 = this.finalSigma(this.SO1());
    }

    void setO2Output() {
        sigmaSO2 = this.finalSigma(this.SO2());
    }

    void setO3Output() {
        sigmaSO3 = this.finalSigma(this.SO3());
    }

    double getO1Output() {
        return sigmaSO1;
    }

    double getO2Output() {
        return sigmaSO2;
    }

    double getO3Output() {
        return sigmaSO3;
    }


    //classify the values
    String classify(double finSigma1, double finSigma2, double finSigma3) {
        //if sigma value of SO1 is greater than that of SO2 & SO3
        if ((finSigma1 > finSigma2) && (finSigma1 > finSigma3)) {
            //return as Iris Setosa
            this.setObservedOutputName(setosaName);
            observedOutput = getO1Output();
            return setosaName;
        } else if ((finSigma2 > finSigma1) && (finSigma2 > finSigma3)) {
            //else if return as Iris Versicolour
            this.setObservedOutputName(versicolourName);
            observedOutput = getO2Output();
            return versicolourName;
        } else {
            //else return as Iris Virginica
            this.setObservedOutputName(virginicaName);
            observedOutput = getO3Output();
            return virginicaName;
        }
    }

    void comparePlants(double inputSepalLength, double inputSepalWidth,
double inputPetalLength, double inputPetalWidth) {
        setosaPoints = 0;
        versicolourPoints = 0;
        virginicaPoints = 0;


        //Compare sepal lengths
        if (inputSepalLength >= setosaSepalLengthMin && inputSepalLength <=
setosaSepalLengthMax) {
```

```
            setosaPoints++;
            if (inputSepalLength >= versicolourSepalLengthMin) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputSepalLength >= versicolourSepalLengthMin) &&
(inputSepalLength <= versicolourSepalLengthMax)) {
            versicolourPoints++;
            virginicaPoints++;

            if ((inputSepalLength >= versicolourSepalLengthMin) &&
(inputSepalLength <= setosaSepalLengthMax)) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputSepalLength >= virginicaSepalLengthMin) &&
(inputSepalLength <= virginicaSepalLengthMax)) {
            virginicaPoints++;
            if (inputSepalLength >= versicolourSepalLengthMax) {
                virginicaPoints++;
            }
        }


        //compare sepal widths
        if (inputSepalWidth >= setosaSepalWidthMin && inputSepalWidth <=
setosaSepalWidthMax) {
            setosaPoints++;
            if (inputSepalWidth <= versicolourSepalWidthMax) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if (inputSepalWidth >= versicolourSepalWidthMin && inputSepalWidth <=
versicolourSepalWidthMax) {
            versicolourPoints++;
            if (inputPetalWidth >= virginicaSepalWidthMin) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            } else {
                versicolourPoints++;
            }
        }

        if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth <=
virginicaSepalWidthMax) {
            virginicaPoints++;
            if (inputSepalWidth <= versicolourSepalWidthMax) {
```

```
                setosaPoints++;
                virginicaPoints++;
            }
        }

        if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth <=
versicolourSepalWidthMax) {
            virginicaPoints++;
            versicolourPoints++;
        }

        //compare petal lengths
        if ((inputPetalLength >= setosaPetalLengthMin) && (inputPetalLength
<= setosaPetalLengthMax)) {
            setosaPoints++;
        }

        if ((inputPetalLength >= versicolourPetalLengthMin) &&
(inputPetalLength <= versicolourPetalLengthMax)) {
            versicolourPoints++;
            if (inputPetalLength >= virginicaPetalLengthMin) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputPetalLength >= virginicaPetalLengthMin) &&
(inputPetalLength <= virginicaPetalLengthMax)) {
            virginicaPoints++;
            if (inputPetalLength <= versicolourPetalLengthMax) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        //compare petal widths
        if ((inputPetalWidth >= setosaPetalWidthMin) && (inputPetalWidth <=
setosaPetalWidthMax)) {
            setosaPoints++;
        }

        //setosa tie breaker
        if (inputSepalLength >= setosaSepalLengthMin && inputSepalLength <=
setosaSepalLengthMax) {
            if (inputSepalWidth >= setosaSepalWidthMin && inputSepalWidth <=
setosaSepalWidthMax) {
                if (inputPetalLength >= setosaPetalLengthMin &&
inputPetalLength <= setosaPetalLengthMax) {
                    if (inputPetalWidth >= setosaPetalWidthMin &&
inputPetalWidth <= setosaPetalWidthMax) {
                        setosaPoints++;
                    }
                }
            }
        }

        //versicolour tie breaker
```

```
        if (inputSepalLength >= versicolourSepalLengthMin && inputSepalLength
<= versicolourSepalLengthMax) {
            if (inputSepalWidth >= versicolourSepalWidthMin &&
inputSepalWidth <= versicolourSepalWidthMax) {
                if (inputPetalLength >= versicolourPetalLengthMin &&
inputPetalLength <= versicolourPetalLengthMax) {
                    if (inputPetalWidth >= versicolourPetalWidthMin &&
inputPetalWidth <= versicolourPetalWidthMax) {
                        versicolourPoints++;
                    }
                }
            }
        }

        //virginica tie breaker
        if (inputSepalLength >= virginicaSepalLengthMin && inputSepalLength
<= virginicaSepalLengthMax) {
            if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth
<= virginicaSepalWidthMax) {
                if (inputPetalLength >= virginicaPetalLengthMin &&
inputPetalLength <= virginicaPetalLengthMax) {
                    if (inputPetalWidth >= virginicaPetalWidthMin &&
inputPetalWidth <= virginicaPetalWidthMax) {
                        virginicaPoints++;
                    }
                }
            }
        }

        if ((inputPetalWidth >= versicolourPetalWidthMin) && (inputPetalWidth
<= versicolourPetalWidthMax)) {
            versicolourPoints++;
            if (inputPetalWidth >= virginicaPetalWidthMin) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputPetalWidth >= virginicaPetalWidthMin) && (inputPetalWidth
<= virginicaPetalWidthMax)) {
            virginicaPoints++;
            if (inputPetalWidth <= versicolourPetalWidthMax) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if (setosaPoints > versicolourPoints && setosaPoints >
virginicaPoints) {
            targetOutput = getO1Output();
            setosaTargetOutput = 1;
            versicolourTargetOutput = 0;
            virginicaTargetOutput = 0;
            targetOutputName = setosaName;
        } else if (versicolourPoints > setosaPoints && versicolourPoints >
virginicaPoints) {
            targetOutput = getO2Output();
```

```
            setosaTargetOutput = 0;
            versicolourTargetOutput = 1;
            virginicaTargetOutput = 0;
            targetOutputName = versicolourName;
        } else {
            targetOutput = getO3Output();
            setosaTargetOutput = 0;
            versicolourTargetOutput = 0;
            virginicaTargetOutput = 1;
            targetOutputName = virginicaName;
        }
    }

    //return the learning rate
    double learningRate() {
        return 0.1;
    }


    //step 3: error calculation between actual output and desired output
    static double outputLayerErrorCalculation(double outputLayerNeuronResult,
double targetOutput) {
        return outputLayerNeuronResult * (1 - outputLayerNeuronResult) *
(targetOutput - outputLayerNeuronResult);
    }

    void setO1OutputLayerError() {
        O1OutputLayerError = outputLayerErrorCalculation(getO1Output(),
setosaTargetOutput);
    }

    void setO2OutputLayerError() {
        O2OutputLayerError = outputLayerErrorCalculation(getO2Output(),
versicolourTargetOutput);
    }

    void setO3OutputLayerError() {
        O3OutputLayerError = outputLayerErrorCalculation(getO3Output(),
virginicaTargetOutput);
    }

    double getO1OutputLayerError() {
        return O1OutputLayerError;
    }

    double getO2OutputLayerError() {
        return O2OutputLayerError;
    }

    double getO3OutputLayerError() {
        return O3OutputLayerError;
    }

    //step 3.5: calculate DeltaIK (between hidden & output layers)
    double DeltaIK(double hiddenLayerNeuron, double OxOutputLayerError) {
        return learningRate() * OxOutputLayerError * hiddenLayerNeuron;
    }
```

```
    //step 4: Update weights between hidden and output layers
    void updateSOxWeights() {
        setO1OutputLayerError();
        setO2OutputLayerError();
        setO3OutputLayerError();
        SO1weight1 += this.DeltaIK(sigmaSH1(), getO1OutputLayerError());
        SO1weight2 += this.DeltaIK(sigmaSH2(), getO1OutputLayerError());
        SO2weight1 += this.DeltaIK(sigmaSH1(), getO2OutputLayerError());
        SO2weight2 += this.DeltaIK(sigmaSH2(), getO2OutputLayerError());
        SO3weight1 += this.DeltaIK(sigmaSH1(), getO3OutputLayerError());
        SO3weight2 += this.DeltaIK(sigmaSH2(), getO3OutputLayerError());

    }

    //step 5: calculate error between input and hidden layers
    double hiddenLayerErrorCalculation(double sigmaSHxOutput, double
summationx) {
        double errorCalc = sigmaSHxOutput * (1 - sigmaSHxOutput);
        return errorCalc * summationx;
    }

    void setH1HiddenLayerError() {
        H1OutputLayerError = hiddenLayerErrorCalculation(sigmaSH1,
summation1);
    }

    void setH2HiddenLayerError() {
        H2OutputLayerError = hiddenLayerErrorCalculation(sigmaSH2,
summation2);
    }

    double getH1OutputLayerError() {
        return H1OutputLayerError;
    }

    double getH2OutputLayerError() {
        return H2OutputLayerError;
    }

    //step 6: Update weights between input and hidden layers
    void updateSHxWeights() {
        setH1HiddenLayerError();
        setH2HiddenLayerError();
        SH1weight0 += this.DeltaLI(BIAS_NEURON, getH1OutputLayerError());
        SH1weight1 += this.DeltaLI(getX1(), getH1OutputLayerError());
        SH1weight2 += this.DeltaLI(getX2(), getH1OutputLayerError());
        SH1weight3 += this.DeltaLI(getX3(), getH1OutputLayerError());
        SH1weight4 += this.DeltaLI(getX4(), getH1OutputLayerError());
        SH2weight0 += this.DeltaLI(BIAS_NEURON, getH2OutputLayerError());
        SH2weight1 += this.DeltaLI(getX1(), getH2OutputLayerError());
        SH2weight2 += this.DeltaLI(getX2(), getH2OutputLayerError());
        SH2weight3 += this.DeltaLI(getX3(), getH2OutputLayerError());
        SH2weight4 += this.DeltaLI(getX4(), getH2OutputLayerError());
    }

    //step 6.5: calculate DeltaLI (between input & hidden layers)
```

```
    double DeltaLI(double inputLayerNeuron, double HxOutputLayerError) {
        return learningRate() * HxOutputLayerError * inputLayerNeuron;
    }

    void setSummations(){
        double SH1SO1Error = getH1OutputLayerError() * SO1weight1;
        double SH1SO2Error = getH1OutputLayerError() * SO2weight1;
        double SH1SO3Error = getH1OutputLayerError() * SO3weight1;
        double SH2SO1Error = getH2OutputLayerError() * SO1weight2;
        double SH2SO2Error = getH2OutputLayerError() * SO2weight2;
        double SH2SO3Error = getH2OutputLayerError() * SO3weight2;
        summation1 = SH1SO1Error + SH1SO2Error + SH1SO3Error;
        summation2 = SH2SO1Error + SH2SO2Error + SH2SO3Error;
    }

    @GetMapping("/train-ann")
    void trainPerceptron(String trainDataUrl) {
        int trainingNum = 0;
        this.loadDataset("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTraining.data");
        String plantName;

        //for all available data in the testing file
        for (int i = 0; i < irisLines.size(); i++) {
            //individually set values from file
            //sepal length
            input[0] = Double.parseDouble(irisValues.get(i)[0]);
            //sepal width
            input[1] = Double.parseDouble(irisValues.get(i)[1]);
            //petal length
            input[2] = Double.parseDouble(irisValues.get(i)[2]);
            //petal width
            input[3] = Double.parseDouble(irisValues.get(i)[3]);
            //plant name
            plantName = irisValues.get(i)[4];

            System.out.println("-------- 5-2-3 ANN IRIS DATASET TRAINING " +
trainingNum + "-------");
            int epochNum = 0;

            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));

                //output current epoch results
                //calculate the weighted sums into the output layer
                //finally the output from the output layer
                setO1Output();
                setO2Output();
                setO3Output();
                //compare values
                this.comparePlants(input[0], input[1], input[2], input[3]);

                this.updateSOxWeights();

                //set summations
                this.setSummations();
```

```
                this.updateSHxWeights();

                System.out.println("Final values are: SO1 " + getO1Output() +
" SO2 " + getO2Output() + " SO3 " + getO3Output());

                //output current epoch results
                this.classify(getO1Output(), getO2Output(), getO3Output());

                System.out.println("Observed output: " + observedOutput + " |
Category: " + this.getObservedOutputName());
                System.out.println("Target output: " + targetOutput + " |
Category: " + this.targetOutputName);
                System.out.println("input values are: " + input[0] + ", " +
input[1] + ", " + input[2] + ", " + input[3] + ", " + plantName + "\n");

                //increment epoch number
                epochNum++;
                //continue loop until values are equal
            } while (!Objects.equals(targetOutputName, observedOutputName) &&
!Objects.equals(plantName, observedOutputName));
            trainingNum++;
        }
    }

    //execute the perceptron
    @GetMapping("/test-ann")
    String executePerceptron(String testDataUrl) {
        int iterationNum = 0;
        this.loadDataset("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTesting.data");
        String plantName;
        correctlyClassified = 0;

        //for all available data in the testing file
        for (int i = 0; i < irisLines.size(); i++) {
            //individually set values from file
            //sepal length
            input[0] = Double.parseDouble(irisValues.get(i)[0]);
            //sepal width
            input[1] = Double.parseDouble(irisValues.get(i)[1]);
            //petal length
            input[2] = Double.parseDouble(irisValues.get(i)[2]);
            //petal width
            input[3] = Double.parseDouble(irisValues.get(i)[3]);
            //plant name
            plantName = irisValues.get(i)[4];

            System.out.println("--------- 5-2-3 ANN IRIS DATASET OFFICIAL RUN
" + iterationNum + "---------");
            //set epoch to zero for looping
            int epochNum = 0;

            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));
```

```
                //output current epoch results
                //calculate the weighted sums into the output layer
                //finally the output from the output layer
                setO1Output();
                setO2Output();
                setO3Output();

                //compare values
                this.comparePlants(input[0], input[1], input[2], input[3]);

                //step 3: calculate the error between actual output and
desired output
                //See outputLayerErrorCalculation function

                //step 4: adjust each weight in the connections between
output and hidden layers
                this.updateSOxWeights();

                //step 5: calculate error between input and hidden layer
(part of step 6's function)
                //set summations
                this.setSummations();

                //step 6: adjust each weight in the connections between
hidden and input layers
                this.updateSHxWeights();

                //output from output layer
                System.out.println("Final values are: SO1 " + getO1Output() +
" SO2 " + getO2Output() + " SO3 " + getO3Output());

                //classify the values
                classify(getO1Output(), getO2Output(), getO3Output());

                //output results
                System.out.println("Observed output: " + observedOutput + " |
Category: " + this.getObservedOutputName());
                System.out.println("Target output: " + targetOutput + " |
Category: " + this.targetOutputName);
                System.out.println("input values are: " + input[0] + ", " +
input[1] + ", " + input[2] + ", " + input[3] + ", " + plantName + "\n");

            /* System.out.println("Setosa points " + setosaPoints);
             System.out.println("Versicolour points " + versicolourPoints);
             System.out.println("Virginica points " + virginicaPoints);*/
                //increment epoch number
                epochNum++;

                //Calculate Network Error
                double O1NetError = differenceSquared(setosaTargetOutput,
getO1Output());
                double O2NetError =
differenceSquared(versicolourTargetOutput, getO2Output());
                double O3NetError =
differenceSquared(versicolourTargetOutput, getO3Output());
                OutputLayerNetError = O1NetError + O2NetError + O3NetError;
                //continue loop until values are equal
```

```
            } while (!Objects.equals(targetOutputName, observedOutputName));

            //Calculate Network Accuracy
            if (plantName.equals(observedOutputName) &&
plantName.equals(targetOutputName)){
                correctlyClassified++;
            }
            this.networkAccuracy = (correctlyClassified /
irisLines.size())*100;
            this.networkError += OutputLayerNetError;
            iterationNum++;
        }
        return "Accuracy of the 5-2-3 ANN is: " + networkAccuracy+ "%";
    }

    void loadDataset(String url) {
        irisLines.clear();
        irisValues.clear();
        //receive input from file
        try {
            //File irisData = new File("/Users/boost/IdeaProjects/AI Test
2/src/iris/iris.data");

            File irisData = new File(url);
            Scanner scanner = new Scanner(irisData);
            while (scanner.hasNext()) {
                String next = scanner.next();
                //prints the data for testing
                //System.out.println(next);
                irisLines.add(next);
            }

            scanner.close();
        } catch (Exception e) {
            System.out.println("error " + e);
        }

        //assign all values to a list of arrays for value separation
        for (int i = 0; i < irisLines.size(); i++) {
            irisValues.add(irisLines.get(i).split(","));
        }


        //test output
        //System.out.println("Display test " + irisLines.get(0) + " and " +
irisValues.get(0)[0]);

        //Iris Setosa Samples ( 0 - 50)
        for (int samples = 0; samples < irisLines.size() / 3; samples++) {
            //Sepal Length of sample
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            setosaSepalLengthAvg += sepalLength;

            //get maximum sepal length for the setosa
            if (setosaSepalLengthMax < sepalLength) {
                setosaSepalLengthMax = sepalLength;
```

```
            }

            //get minimum sepal length for the setosa
            if (setosaSepalLengthMin > sepalLength) {
                setosaSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            setosaSepalWidthAvg += sepalWidth;

            //get maximum sepal width for the setosa
            if (setosaSepalWidthMax < sepalWidth) {
                setosaSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the setosa
            if (setosaSepalWidthMin > sepalWidth) {
                setosaSepalWidthMin = sepalWidth;
            }

            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            setosaPetalLengthAvg += petalLength;

            //max petal length of setosa
            if (setosaPetalLengthMax < petalLength) {
                setosaPetalLengthMax = petalLength;
            }

            //min petal length of setosa
            if (setosaPetalLengthMin > petalLength) {
                setosaPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            setosaPetalWidthAvg += petalWidth;

            //max petal width of setosa
            if (setosaPetalWidthMax < petalWidth) {
                setosaPetalWidthMax = petalWidth;
            }

            //min petal width of setosa
            if (setosaPetalWidthMin > petalWidth) {
                setosaPetalWidthMin = petalWidth;
            }

            //Class Name of sample
            setosaName = irisValues.get(samples)[4];

        }
        setosaSepalLengthAvg = setosaSepalLengthAvg / (irisLines.size() / 3);
```

```
        setosaSepalWidthAvg = setosaSepalWidthAvg / (irisLines.size() / 3);
        setosaPetalLengthAvg = setosaPetalLengthAvg / (irisLines.size() / 3);
        setosaPetalWidthAvg = setosaPetalWidthAvg / (irisLines.size() / 3);
        setosaTotalAvg = (setosaPetalLengthAvg + setosaPetalWidthAvg +
setosaSepalLengthAvg + setosaSepalWidthAvg) / 4;

        System.out.println("                        Min   Max   Mean");
        System.out.println("setosa sepal length:  " + setosaSepalLengthMin +
"   " + setosaSepalLengthMax + "   " + setosaSepalLengthAvg);
        System.out.println("setosa sepal width:   " + setosaSepalWidthMin + "
" + setosaSepalWidthMax + "   " + setosaSepalWidthAvg);
        System.out.println("setosa petal length:  " + setosaPetalLengthMin +
"   " + setosaPetalLengthMax + "   " + setosaPetalLengthAvg);
        System.out.println("setosa petal width:   " + setosaPetalWidthMin + "
" + setosaPetalWidthMax + "   " + setosaPetalWidthAvg);
        System.out.println("setosa total average: " + setosaTotalAvg + "\n");

        //Iris Versicolour Samples (50 - 100)
        for (int samples = irisLines.size() / 3; samples < (irisLines.size()
/ 3) * 2; samples++) {
            //Sepal Length of sample
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            versicolourSepalLengthAvg += sepalLength;

            //get maximum sepal length for the versicolour
            if (versicolourSepalLengthMax < sepalLength) {
                versicolourSepalLengthMax = sepalLength;
            }

            //get minimum sepal length for the versicolour
            if (versicolourSepalLengthMin > sepalLength) {
                versicolourSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            versicolourSepalWidthAvg += sepalWidth;

            //get maximum sepal width for the versicolour
            if (versicolourSepalWidthMax < sepalWidth) {
                versicolourSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the versicolour
            if (versicolourSepalWidthMin > sepalWidth) {
                versicolourSepalWidthMin = sepalWidth;
            }

            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            versicolourPetalLengthAvg += petalLength;

            //max petal length of versicolour
            if (versicolourPetalLengthMax < petalLength) {
```

```
                versicolourPetalLengthMax = petalLength;
            }

            //min petal length of versicolour
            if (versicolourPetalLengthMin > petalLength) {
                versicolourPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            versicolourPetalWidthAvg += petalWidth;

            //max petal width of versicolour
            if (versicolourPetalWidthMax < petalWidth) {
                versicolourPetalWidthMax = petalWidth;
            }

            //min petal width of versicolour
            if (versicolourPetalWidthMin > petalWidth) {
                versicolourPetalWidthMin = petalWidth;
            }

            //Class Name of sample
            versicolourName = irisValues.get(samples)[4];

        }
        versicolourSepalLengthAvg = versicolourSepalLengthAvg /
(irisLines.size() / 3);
        versicolourSepalWidthAvg = versicolourSepalWidthAvg /
(irisLines.size() / 3);
        versicolourPetalLengthAvg = versicolourPetalLengthAvg /
(irisLines.size() / 3);
        versicolourPetalWidthAvg = versicolourPetalWidthAvg /
(irisLines.size() / 3);
        versicolourTotalAvg = (versicolourPetalLengthAvg +
versicolourPetalWidthAvg + versicolourSepalLengthAvg +
versicolourSepalWidthAvg) / 4;

        System.out.println("                              Min   Max   Mean");
        System.out.println("versicolour sepal length:  " +
versicolourSepalLengthMin + "   " + versicolourSepalLengthMax + "   " +
versicolourSepalLengthAvg);
        System.out.println("versicolour sepal width:   " +
versicolourSepalWidthMin + "   " + versicolourSepalWidthMax + "   " +
versicolourSepalWidthAvg);
        System.out.println("versicolour petal length:  " +
versicolourPetalLengthMin + "   " + versicolourPetalLengthMax + "   " +
versicolourPetalLengthAvg);
        System.out.println("versicolour petal width:   " +
versicolourPetalWidthMin + "   " + versicolourPetalWidthMax + "   " +
versicolourPetalWidthAvg);
        System.out.println("versicolour total average: " +
versicolourTotalAvg + "\n");

        //Iris virginica Samples (100 - 150)
        for (int samples = (irisLines.size() / 3) * 2; samples <
```

```
(irisLines.size() / 3) * 3; samples++) {
            //Sepal Length of sample
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            virginicaSepalLengthAvg += sepalLength;

            //get maximum sepal length for the virginica
            if (virginicaSepalLengthMax < sepalLength) {
                virginicaSepalLengthMax = sepalLength;
            }

            //get minimum sepal length for the virginica
            if (virginicaSepalLengthMin > sepalLength) {
                virginicaSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            virginicaSepalWidthAvg += sepalWidth;

            //get maximum sepal width for the virginica
            if (virginicaSepalWidthMax < sepalWidth) {
                virginicaSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the virginica
            if (virginicaSepalWidthMin > sepalWidth) {
                virginicaSepalWidthMin = sepalWidth;
            }

            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            virginicaPetalLengthAvg += petalLength;

            //max petal length of virginica
            if (virginicaPetalLengthMax < petalLength) {
                virginicaPetalLengthMax = petalLength;
            }

            //min petal length of virginica
            if (virginicaPetalLengthMin > petalLength) {
                virginicaPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            virginicaPetalWidthAvg += petalWidth;

            //max petal width of virginica
            if (virginicaPetalWidthMax < petalWidth) {
                virginicaPetalWidthMax = petalWidth;
            }

            //min petal width of virginica
```

```
            if (virginicaPetalWidthMin > petalWidth) {
                virginicaPetalWidthMin = petalWidth;
            }

            //Class Name of sample
            virginicaName = irisValues.get(samples)[4];


        }
        virginicaSepalLengthAvg = virginicaSepalLengthAvg / (irisLines.size()
/ 3);
        virginicaSepalWidthAvg = virginicaSepalWidthAvg / (irisLines.size() /
3);
        virginicaPetalLengthAvg = virginicaPetalLengthAvg / (irisLines.size()
/ 3);
        virginicaPetalWidthAvg = virginicaPetalWidthAvg / (irisLines.size() /
3);
        virginicaTotalAvg = (virginicaPetalLengthAvg + virginicaPetalWidthAvg
+ virginicaSepalLengthAvg + virginicaSepalWidthAvg) / 4;

        System.out.println("                          Min   Max   Mean");
        System.out.println("virginica sepal length:  " +
virginicaSepalLengthMin + "   " + virginicaSepalLengthMax + "   " +
virginicaSepalLengthAvg);
        System.out.println("virginica sepal width:   " +
virginicaSepalWidthMin + "   " + virginicaSepalWidthMax + "   " +
virginicaSepalWidthAvg);
        System.out.println("virginica petal length:  " +
virginicaPetalLengthMin + "   " + virginicaPetalLengthMax + "   " +
virginicaPetalLengthAvg);
        System.out.println("virginica petal width:   " +
virginicaPetalWidthMin + "   " + virginicaPetalWidthMax + "   " +
virginicaPetalWidthAvg);
        System.out.println("virginica total average: " + virginicaTotalAvg +
"\n");
    }

    double differenceSquared(double plantTargetOut, double
outputLayerNeuronResult) {
        double sumOutputError = Math.pow((plantTargetOut -
outputLayerNeuronResult), 2);
        return sumOutputError;
    }


    public static void main(String[] args) {
        //create necessary values
        Perceptron523 myPerceptron = new Perceptron523();

        //train the ANN
        //myPerceptron.trainPerceptron("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTraining.data");

        //accept values manually for input (for testing & debugging)
        //myPerceptron.inputPerceptron();

        //execute the ANN program
```

```
        /*myPerceptron.executePerceptron("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTesting.data");
        System.out.println("Network Error: " + (0.5 *
myPerceptron.networkError));
        System.out.println("Correctly Classified: " +
myPerceptron.correctlyClassified);
        System.out.println("Network Accuracy: " +
myPerceptron.networkAccuracy + "%");*/
    }
}
```

5-6-3 ANN Code:

During the building of part 2 of test 2, I found that all the knowledge I'd gained from part 1 felt more natural to me. All I had to do for part 2 was increase the number of hidden layer neurons and make a few adjustments to existing functions to create the 5-6-3 ANN from my 5-2-3 ANN code. I made few unique additions to the 5-6-3 ANN as everything was already functional. I find the accuracy of the dataset while using this ANN to be equal to that of the 5-2-3. I assume that is because of the overlapping nature of the dataset itself.

```
package com.example.aitest2ui;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.io.File;
import java.util.*;
import java.lang.*;

@CrossOrigin(origins = "http://localhost:3000/")
@RestController
@RequestMapping(value = "/ANN563")
public class Perceptron563 {
    Scanner testInput = new Scanner(System.in);

    //initialize values
    static List<String> irisLines = new ArrayList<>();
    static List<String[]> irisValues = new ArrayList<>();
    static String setosaName = "";
    static double setosaSepalLengthAvg = 0, setosaSepalWidthAvg = 0,
setosaPetalLengthAvg = 0, setosaPetalWidthAvg = 0, setosaTotalAvg = 0;
    static double setosaSepalLengthMin = 10, setosaSepalLengthMax = 0,
setosaSepalWidthMin = 10, setosaSepalWidthMax = 0, setosaPetalLengthMax = 0,
setosaPetalLengthMin = 10, setosaPetalWidthMax = 0, setosaPetalWidthMin = 10;
    static String versicolourName = "";
    static double versicolourSepalLengthAvg = 0, versicolourSepalWidthAvg =
0, versicolourPetalLengthAvg = 0, versicolourPetalWidthAvg = 0,
versicolourTotalAvg = 0;
    static double versicolourSepalLengthMin = 10, versicolourSepalLengthMax =
0, versicolourSepalWidthMin = 10, versicolourSepalWidthMax = 0,
versicolourPetalLengthMax = 0, versicolourPetalLengthMin = 10,
versicolourPetalWidthMax = 0, versicolourPetalWidthMin = 10;

    static String virginicaName = "";
    static double virginicaSepalLengthAvg = 0, virginicaSepalWidthAvg = 0,
```

```
virginicaPetalLengthAvg = 0, virginicaPetalWidthAvg = 0, virginicaTotalAvg =
0;
    static double virginicaSepalLengthMin = 10, virginicaSepalLengthMax = 0,
virginicaSepalWidthMin = 10, virginicaSepalWidthMax = 0,
virginicaPetalLengthMax = 0, virginicaPetalLengthMin = 10,
virginicaPetalWidthMax = 0, virginicaPetalWidthMin = 10;
    static double setosaPoints = 0;
    static double versicolourPoints = 0;
    static double virginicaPoints = 0;
    //array to store input positions
    private final double[] input = new double[4];
    private static double targetOutput, observedOutput;
    private final int BIAS_NEURON = 1;
    private double x1;
    private double x2;
    private double x3;
    private double x4;
    private String observedOutputName;
    private String targetOutputName;

    //Weights for BackPropagation
    //SOx being the output layer neuron and weight x being the hidden layer
neuron
    static double SO1weight1 = (Math.random() - .5);
    static double SO1weight2 = (Math.random() - .5);
    static double SO1weight3 = (Math.random() - .5);
    static double SO1weight4 = (Math.random() - .5);
    static double SO1weight5 = (Math.random() - .5);
    static double SO1weight6 = (Math.random() - .5);
    static double SO2weight1 = (Math.random() - .5);
    static double SO2weight2 = (Math.random() - .5);
    static double SO2weight3 = (Math.random() - .5);
    static double SO2weight4 = (Math.random() - .5);
    static double SO2weight5 = (Math.random() - .5);
    static double SO2weight6 = (Math.random() - .5);
    static double SO3weight1 = (Math.random() - .5);
    static double SO3weight2 = (Math.random() - .5);
    static double SO3weight3 = (Math.random() - .5);
    static double SO3weight4 = (Math.random() - .5);
    static double SO3weight5 = (Math.random() - .5);
    static double SO3weight6 = (Math.random() - .5);

    //SHx being the hidden layer neuron and weight x being the input layer
neuron
    //SH1weights
    static double SH1weight0 = (Math.random() - .5);
    static double SH1weight1 = (Math.random() - .5);
    static double SH1weight2 = (Math.random() - .5);
    static double SH1weight3 = (Math.random() - .5);
    static double SH1weight4 = (Math.random() - .5);
    //SH2weights
    static double SH2weight0 = (Math.random() - .5);
    static double SH2weight1 = (Math.random() - .5);
    static double SH2weight2 = (Math.random() - .5);
    static double SH2weight3 = (Math.random() - .5);
    static double SH2weight4 = (Math.random() - .5);
```

```
    //SH3weights
    static double SH3weight0 = (Math.random() - .5);
    static double SH3weight1 = (Math.random() - .5);
    static double SH3weight2 = (Math.random() - .5);
    static double SH3weight3 = (Math.random() - .5);
    static double SH3weight4 = (Math.random() - .5);

    //SH4weights
    static double SH4weight0 = (Math.random() - .5);
    static double SH4weight1 = (Math.random() - .5);
    static double SH4weight2 = (Math.random() - .5);
    static double SH4weight3 = (Math.random() - .5);
    static double SH4weight4 = (Math.random() - .5);

    //SH5weights
    static double SH5weight0 = (Math.random() - .5);
    static double SH5weight1 = (Math.random() - .5);
    static double SH5weight2 = (Math.random() - .5);
    static double SH5weight3 = (Math.random() - .5);
    static double SH5weight4 = (Math.random() - .5);

    //SH6weights
    static double SH6weight0 = (Math.random() - .5);
    static double SH6weight1 = (Math.random() - .5);
    static double SH6weight2 = (Math.random() - .5);
    static double SH6weight3 = (Math.random() - .5);
    static double SH6weight4 = (Math.random() - .5);

    static double summation1 = 0, summation2 = 0, summation3 = 0, summation4
= 0, summation5 = 0, summation6 = 0;
    double sigmaSO1, sigmaSO2, sigmaSO3, sigmaSH1, sigmaSH2, sigmaSH3,
sigmaSH4, sigmaSH5, sigmaSH6;
    double setosaTargetOutput, versicolourTargetOutput,
virginicaTargetOutput;
    double O1OutputLayerError, O2OutputLayerError, O3OutputLayerError;
    double H1OutputLayerError, H2OutputLayerError, H3OutputLayerError,
H4OutputLayerError, H5OutputLayerError, H6OutputLayerError;
    double networkError, OutputLayerNetError, correctlyClassified,
networkAccuracy;

    //default constructor
    Perceptron563() {
    }

    //get observed output name
    String getObservedOutputName() {
        return this.observedOutputName;
    }

    //set observed output name
    void setObservedOutputName(String newOutputName) {
        this.observedOutputName = newOutputName;
    }

    public double getX1() {
        return x1;
    }
```

```
    public void setX1(double x1) {
        this.x1 = x1;
    }

    public double getX2() {
        return x2;
    }

    public void setX2(double x2) {
        this.x2 = x2;
    }

    public double getX3() {
        return x3;
    }

    public void setX3(double x3) {
        this.x3 = x3;
    }

    public double getX4() {
        return x4;
    }

    public void setX4(double x4) {
        this.x4 = x4;
    }

    //initialize x values manually
    String inputPerceptron() {
        //receive X input values
        System.out.println("Sepal Length (cm):");
        setX1(testInput.nextDouble());
        System.out.println("Sepal Width (cm):");
        setX2(testInput.nextDouble());
        System.out.println("Petal Length (cm):");
        setX3(testInput.nextDouble());
        System.out.println("Petal Width (cm):");
        setX4(testInput.nextDouble());

        //accept values for each pixel
        setX1(x1);
        setX2(x2);
        setX3(x3);
        setX4(x4);

        //assign each value to an array position
        input[0] = getX1();
        input[1] = getX2();
        input[2] = getX3();
        input[3] = getX4();

        return "The input values are " + x1 + ", " + x2 + ", " + x3 + ", " +
x4 + ".";
    }
```

```
    //Hidden layer 1 calculations
    double SH1() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH1weight0) + (this.x1 * SH1weight1) +
(this.x2 * SH1weight2) + (this.x3 * SH1weight3) + (this.x4 * SH1weight4));
    }

    //Hidden layer 2 calculations
    double SH2() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH2weight0) + (this.x1 * SH2weight1) +
(this.x2 * SH2weight2) + (this.x3 * SH2weight3) + (this.x4 * SH2weight4));
    }

    //Hidden layer 3 calculations
    double SH3() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH3weight0) + (this.x1 * SH3weight1) +
(this.x2 * SH3weight2) + (this.x3 * SH3weight3) + (this.x4 * SH3weight4));
    }

    //Hidden layer 4 calculations
    double SH4() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH4weight0) + (this.x1 * SH4weight1) +
(this.x2 * SH4weight2) + (this.x3 * SH4weight3) + (this.x4 * SH4weight4));
    }

    //Hidden layer 5 calculations
    double SH5() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH5weight0) + (this.x1 * SH5weight1) +
(this.x2 * SH5weight2) + (this.x3 * SH5weight3) + (this.x4 * SH5weight4));
    }

    //Hidden layer 6 calculations
    double SH6() {
        //set new values for arbitrary weights
        return ((BIAS_NEURON * SH6weight0) + (this.x1 * SH6weight1) +
(this.x2 * SH6weight2) + (this.x3 * SH6weight3) + (this.x4 * SH6weight4));
    }

    //Sigma calculation to output from hidden layer
    double sigmaS(double SHx) {
        //used -1 rather than -SHx to avoid error
        return (1 / (1 + Math.exp(-1 * SHx)));
    }

    double sigmaSH1() {
        sigmaSH1 = sigmaS(SH1());
        return sigmaSH1;
    }

    double sigmaSH2() {
        sigmaSH2 = sigmaS(SH2());
        return sigmaSH2;
    }
```

```
    double sigmaSH3() {
        sigmaSH3 = sigmaS(SH3());
        return sigmaSH3;
    }

    double sigmaSH4() {
        sigmaSH4 = sigmaS(SH4());
        return sigmaSH4;
    }

    double sigmaSH5() {
        sigmaSH5 = sigmaS(SH5());
        return sigmaSH5;
    }

    double sigmaSH6() {
        sigmaSH6 = sigmaS(SH6());
        return sigmaSH6;
    }

    //Calculate weighted sums into the output layer (arbitrary weights)
    double SO1() {
        double SO1;
        SO1 = ((SO1weight1 * sigmaSH1()) + (SO1weight2 * sigmaSH2()) +
(SO1weight3 * sigmaSH3()) + (SO1weight4 * sigmaSH4()) + (SO1weight5 *
sigmaSH5()) + (SO1weight6 * sigmaSH6()));
        return SO1;
    }

    double SO2() {
        double SO2;
        SO2 = ((SO2weight1 * sigmaSH1()) + (SO2weight2 * sigmaSH2()) +
(SO2weight3 * sigmaSH3()) + (SO2weight4 * sigmaSH4()) + (SO2weight5 *
sigmaSH5()) + (SO2weight6 * sigmaSH6()));
        return SO2;
    }

    double SO3() {
        double SO3;
        SO3 = ((SO3weight1 * sigmaSH1()) + (SO3weight2 * sigmaSH2()) +
(SO3weight3 * sigmaSH3()) + (SO3weight4 * sigmaSH4()) + (SO3weight5 *
sigmaSH5()) + (SO3weight6 * sigmaSH6()));
        return SO3;
    }

    //Calculate the ANN (output from the output layer)
    double finalSigma(double SOx) {
        return sigmaS(SOx);
    }

    void setO1Output() {
        sigmaSO1 = this.finalSigma(this.SO1());
    }

    void setO2Output() {
        sigmaSO2 = this.finalSigma(this.SO2());
```

```
    }

    void setO3Output() {
        sigmaSO3 = this.finalSigma(this.SO3());
    }

    double getO1Output() {
        return sigmaSO1;
    }

    double getO2Output() {
        return sigmaSO2;
    }

    double getO3Output() {
        return sigmaSO3;
    }


    //classify the values
    String classify(double finSigma1, double finSigma2, double finSigma3) {
        //if sigma value of SO1 is greater than that of SO2 & SO3
        if ((finSigma1 > finSigma2) && (finSigma1 > finSigma3)) {
            //return as Iris Setosa
            this.setObservedOutputName(setosaName);
            observedOutput = getO1Output();
            return setosaName;
        } else if ((finSigma2 > finSigma1) && (finSigma2 > finSigma3)) {
            //else if return as Iris Versicolour
            this.setObservedOutputName(versicolourName);
            observedOutput = getO2Output();
            return versicolourName;
        } else {
            //else return as Iris Virginica
            this.setObservedOutputName(virginicaName);
            observedOutput = getO3Output();
            return virginicaName;
        }
    }

    //compares plants and sets targetOutput
    void comparePlants(double inputSepalLength, double inputSepalWidth,
double inputPetalLength, double inputPetalWidth) {
        setosaPoints = 0;
        versicolourPoints = 0;
        virginicaPoints = 0;


        //Compare sepal lengths
        if (inputSepalLength >= setosaSepalLengthMin && inputSepalLength <=
setosaSepalLengthMax) {
            setosaPoints++;
            if (inputSepalLength >= versicolourSepalLengthMin) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
```

```
        }

        if ((inputSepalLength >= versicolourSepalLengthMin) &&
(inputSepalLength <= versicolourSepalLengthMax)) {
            versicolourPoints++;
            virginicaPoints++;

            if ((inputSepalLength >= versicolourSepalLengthMin) &&
(inputSepalLength <= setosaSepalLengthMax)) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputSepalLength >= virginicaSepalLengthMin) &&
(inputSepalLength <= virginicaSepalLengthMax)) {
            virginicaPoints++;
            if (inputSepalLength >= versicolourSepalLengthMax) {
                virginicaPoints++;
            }
        }


        //compare sepal widths
        if (inputSepalWidth >= setosaSepalWidthMin && inputSepalWidth <=
setosaSepalWidthMax) {
            setosaPoints++;
            if (inputSepalWidth <= versicolourSepalWidthMax) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if (inputSepalWidth >= versicolourSepalWidthMin && inputSepalWidth <=
versicolourSepalWidthMax) {
            versicolourPoints++;
            if (inputPetalWidth >= virginicaSepalWidthMin) {
                setosaPoints++;
                versicolourPoints++;
                virginicaPoints++;
            } else {
                versicolourPoints++;
            }
        }

        if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth <=
virginicaSepalWidthMax) {
            virginicaPoints++;
            if (inputSepalWidth <= versicolourSepalWidthMax) {
                setosaPoints++;
                virginicaPoints++;
            }
        }

        if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth <=
```

```
versicolourSepalWidthMax) {
            virginicaPoints++;
            versicolourPoints++;
        }

        //compare petal lengths
        if ((inputPetalLength >= setosaPetalLengthMin) && (inputPetalLength
<= setosaPetalLengthMax)) {
            setosaPoints++;
        }

        if ((inputPetalLength >= versicolourPetalLengthMin) &&
(inputPetalLength <= versicolourPetalLengthMax)) {
            versicolourPoints++;
            if (inputPetalLength >= virginicaPetalLengthMin) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputPetalLength >= virginicaPetalLengthMin) &&
(inputPetalLength <= virginicaPetalLengthMax)) {
            virginicaPoints++;
            if (inputPetalLength <= versicolourPetalLengthMax) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        //compare petal widths
        if ((inputPetalWidth >= setosaPetalWidthMin) && (inputPetalWidth <=
setosaPetalWidthMax)) {
            setosaPoints++;
        }

        //setosa tie breaker
        if (inputSepalLength >= setosaSepalLengthMin && inputSepalLength <=
setosaSepalLengthMax) {
            if (inputSepalWidth >= setosaSepalWidthMin && inputSepalWidth <=
setosaSepalWidthMax) {
                if (inputPetalLength >= setosaPetalLengthMin &&
inputPetalLength <= setosaPetalLengthMax) {
                    if (inputPetalWidth >= setosaPetalWidthMin &&
inputPetalWidth <= setosaPetalWidthMax) {
                        setosaPoints++;
                    }
                }
            }
        }

        //versicolour tie breaker
        if (inputSepalLength >= versicolourSepalLengthMin && inputSepalLength
<= versicolourSepalLengthMax) {
            if (inputSepalWidth >= versicolourSepalWidthMin &&
inputSepalWidth <= versicolourSepalWidthMax) {
                if (inputPetalLength >= versicolourPetalLengthMin &&
inputPetalLength <= versicolourPetalLengthMax) {
```

```
                    if (inputPetalWidth >= versicolourPetalWidthMin &&
inputPetalWidth <= versicolourPetalWidthMax) {
                        versicolourPoints++;
                    }
                }
            }
        }

        //virginica tie breaker
        if (inputSepalLength >= virginicaSepalLengthMin && inputSepalLength
<= virginicaSepalLengthMax) {
            if (inputSepalWidth >= virginicaSepalWidthMin && inputSepalWidth
<= virginicaSepalWidthMax) {
                if (inputPetalLength >= virginicaPetalLengthMin &&
inputPetalLength <= virginicaPetalLengthMax) {
                    if (inputPetalWidth >= virginicaPetalWidthMin &&
inputPetalWidth <= virginicaPetalWidthMax) {
                        virginicaPoints++;
                    }
                }
            }
        }

        if ((inputPetalWidth >= versicolourPetalWidthMin) && (inputPetalWidth
<= versicolourPetalWidthMax)) {
            versicolourPoints++;
            if (inputPetalWidth >= virginicaPetalWidthMin) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if ((inputPetalWidth >= virginicaPetalWidthMin) && (inputPetalWidth
<= virginicaPetalWidthMax)) {
            virginicaPoints++;
            if (inputPetalWidth <= versicolourPetalWidthMax) {
                versicolourPoints++;
                virginicaPoints++;
            }
        }

        if (setosaPoints > versicolourPoints && setosaPoints >
virginicaPoints) {
            targetOutput = getO1Output();
            setosaTargetOutput = 1;
            versicolourTargetOutput = 0;
            virginicaTargetOutput = 0;
            targetOutputName = setosaName;
        } else if (versicolourPoints > setosaPoints && versicolourPoints >
virginicaPoints) {
            targetOutput = getO2Output();
            setosaTargetOutput = 0;
            versicolourTargetOutput = 1;
            virginicaTargetOutput = 0;
            targetOutputName = versicolourName;
        } else {
            targetOutput = getO3Output();
```

```
            setosaTargetOutput = 0;
            versicolourTargetOutput = 0;
            virginicaTargetOutput = 1;
            targetOutputName = virginicaName;
        }
    }

    //return the learning rate
    double learningRate() {
        return 0.1;
    }


    //step 3: error calculation between actual output and desired output
    static double outputLayerErrorCalculation(double outputLayerNeuronResult,
double targetOutput) {
        return outputLayerNeuronResult * (1 - outputLayerNeuronResult) *
(targetOutput - outputLayerNeuronResult);
    }

    void setO1OutputLayerError() {
        O1OutputLayerError = outputLayerErrorCalculation(getO1Output(),
setosaTargetOutput);
    }

    void setO2OutputLayerError() {
        O2OutputLayerError = outputLayerErrorCalculation(getO2Output(),
versicolourTargetOutput);
    }

    void setO3OutputLayerError() {
        O3OutputLayerError = outputLayerErrorCalculation(getO3Output(),
virginicaTargetOutput);
    }

    double getO1OutputLayerError() {
        return O1OutputLayerError;
    }

    double getO2OutputLayerError() {
        return O2OutputLayerError;
    }

    double getO3OutputLayerError() {
        return O3OutputLayerError;
    }

    //step 3.5: calculate DeltaIK (between hidden & output layers)
    double DeltaIK(double hiddenLayerNeuron, double OxOutputLayerError) {
        return learningRate() * OxOutputLayerError * hiddenLayerNeuron;
    }

    //step 4: Update weights between hidden and output layers
    void updateSOxWeights() {
        setO1OutputLayerError();
        setO2OutputLayerError();
        setO3OutputLayerError();
```

```
        SO1weight1 += this.DeltaIK(sigmaSH1(), getO1OutputLayerError());
        SO1weight2 += this.DeltaIK(sigmaSH2(), getO1OutputLayerError());
        SO1weight3 += this.DeltaIK(sigmaSH3(), getO1OutputLayerError());
        SO1weight4 += this.DeltaIK(sigmaSH4(), getO1OutputLayerError());
        SO1weight5 += this.DeltaIK(sigmaSH5(), getO1OutputLayerError());
        SO1weight6 += this.DeltaIK(sigmaSH6(), getO1OutputLayerError());

        SO2weight1 += this.DeltaIK(sigmaSH1(), getO2OutputLayerError());
        SO2weight2 += this.DeltaIK(sigmaSH2(), getO2OutputLayerError());
        SO2weight3 += this.DeltaIK(sigmaSH3(), getO2OutputLayerError());
        SO2weight4 += this.DeltaIK(sigmaSH4(), getO2OutputLayerError());
        SO2weight5 += this.DeltaIK(sigmaSH5(), getO2OutputLayerError());
        SO2weight6 += this.DeltaIK(sigmaSH6(), getO2OutputLayerError());

        SO3weight1 += this.DeltaIK(sigmaSH1(), getO3OutputLayerError());
        SO3weight2 += this.DeltaIK(sigmaSH2(), getO3OutputLayerError());
        SO3weight3 += this.DeltaIK(sigmaSH3(), getO3OutputLayerError());
        SO3weight4 += this.DeltaIK(sigmaSH4(), getO3OutputLayerError());
        SO3weight5 += this.DeltaIK(sigmaSH5(), getO3OutputLayerError());
        SO3weight6 += this.DeltaIK(sigmaSH6(), getO3OutputLayerError());

    }

    //step 5: calculate error between input and hidden layers
    double hiddenLayerErrorCalculation(double sigmaSHxOutput, double
summationx) {
        double errorCalc = sigmaSHxOutput * (1 - sigmaSHxOutput);
        return errorCalc * summationx;
    }

    void setH1HiddenLayerError() {
        H1OutputLayerError = hiddenLayerErrorCalculation(sigmaSH1,
summation1);
    }

    void setH2HiddenLayerError() {
        H2OutputLayerError = hiddenLayerErrorCalculation(sigmaSH2,
summation2);
    }

    void setH3HiddenLayerError() {
        H3OutputLayerError = hiddenLayerErrorCalculation(sigmaSH3,
summation3);
    }

    void setH4HiddenLayerError() {
        H4OutputLayerError = hiddenLayerErrorCalculation(sigmaSH4,
summation4);
    }

    void setH5HiddenLayerError() {
        H5OutputLayerError = hiddenLayerErrorCalculation(sigmaSH5,
summation5);
    }

    void setH6HiddenLayerError() {
        H6OutputLayerError = hiddenLayerErrorCalculation(sigmaSH6,
```

```
summation6);
    }

    double getH1OutputLayerError() {
        return H1OutputLayerError;
    }

    double getH2OutputLayerError() {
        return H2OutputLayerError;
    }

    double getH3OutputLayerError() {
        return H3OutputLayerError;
    }

    double getH4OutputLayerError() {
        return H4OutputLayerError;
    }

    double getH5OutputLayerError() {
        return H5OutputLayerError;
    }

    double getH6OutputLayerError() {
        return H6OutputLayerError;
    }

    //step 6: Update weights between input and hidden layers
    void updateSHxWeights() {
        setH1HiddenLayerError();
        setH2HiddenLayerError();
        setH3HiddenLayerError();
        setH4HiddenLayerError();
        setH5HiddenLayerError();
        setH6HiddenLayerError();
        SH1weight0 += this.DeltaLI(BIAS_NEURON, getH1OutputLayerError());
        SH1weight1 += this.DeltaLI(getX1(), getH1OutputLayerError());
        SH1weight2 += this.DeltaLI(getX2(), getH1OutputLayerError());
        SH1weight3 += this.DeltaLI(getX3(), getH1OutputLayerError());
        SH1weight4 += this.DeltaLI(getX4(), getH1OutputLayerError());

        SH2weight0 += this.DeltaLI(BIAS_NEURON, getH2OutputLayerError());
        SH2weight1 += this.DeltaLI(getX1(), getH2OutputLayerError());
        SH2weight2 += this.DeltaLI(getX2(), getH2OutputLayerError());
        SH2weight3 += this.DeltaLI(getX3(), getH2OutputLayerError());
        SH2weight4 += this.DeltaLI(getX4(), getH2OutputLayerError());

        SH3weight0 += this.DeltaLI(BIAS_NEURON, getH3OutputLayerError());
        SH3weight1 += this.DeltaLI(getX1(), getH3OutputLayerError());
        SH3weight2 += this.DeltaLI(getX2(), getH3OutputLayerError());
        SH3weight3 += this.DeltaLI(getX3(), getH3OutputLayerError());
        SH3weight4 += this.DeltaLI(getX4(), getH3OutputLayerError());

        SH4weight0 += this.DeltaLI(BIAS_NEURON, getH4OutputLayerError());
        SH4weight1 += this.DeltaLI(getX1(), getH4OutputLayerError());
        SH4weight2 += this.DeltaLI(getX2(), getH4OutputLayerError());
        SH4weight3 += this.DeltaLI(getX3(), getH4OutputLayerError());
```

```
        SH4weight4 += this.DeltaLI(getX4(), getH4OutputLayerError());

        SH5weight0 += this.DeltaLI(BIAS_NEURON, getH5OutputLayerError());
        SH5weight1 += this.DeltaLI(getX1(), getH5OutputLayerError());
        SH5weight2 += this.DeltaLI(getX2(), getH5OutputLayerError());
        SH5weight3 += this.DeltaLI(getX3(), getH5OutputLayerError());
        SH5weight4 += this.DeltaLI(getX4(), getH5OutputLayerError());

        SH6weight0 += this.DeltaLI(BIAS_NEURON, getH6OutputLayerError());
        SH6weight1 += this.DeltaLI(getX1(), getH6OutputLayerError());
        SH6weight2 += this.DeltaLI(getX2(), getH6OutputLayerError());
        SH6weight3 += this.DeltaLI(getX3(), getH6OutputLayerError());
        SH6weight4 += this.DeltaLI(getX4(), getH6OutputLayerError());
    }

    //step 6.5: calculate DeltaLI (between input & hidden layers)
    double DeltaLI(double inputLayerNeuron, double HxOutputLayerError) {
        return learningRate() * HxOutputLayerError * inputLayerNeuron;
    }

    void setSummations() {
        double SH1SO1Error = getH1OutputLayerError() * SO1weight1;
        double SH1SO2Error = getH1OutputLayerError() * SO2weight1;
        double SH1SO3Error = getH1OutputLayerError() * SO3weight1;
        summation1 = SH1SO1Error + SH1SO2Error + SH1SO3Error;

        double SH2SO1Error = getH2OutputLayerError() * SO1weight2;
        double SH2SO2Error = getH2OutputLayerError() * SO2weight2;
        double SH2SO3Error = getH2OutputLayerError() * SO3weight2;
        summation2 = SH2SO1Error + SH2SO2Error + SH2SO3Error;

        double SH3SO1Error = getH3OutputLayerError() * SO1weight3;
        double SH3SO2Error = getH3OutputLayerError() * SO2weight3;
        double SH3SO3Error = getH3OutputLayerError() * SO3weight3;
        summation3 = SH3SO1Error + SH3SO2Error + SH3SO3Error;

        double SH4SO1Error = getH4OutputLayerError() * SO1weight4;
        double SH4SO2Error = getH4OutputLayerError() * SO2weight4;
        double SH4SO3Error = getH4OutputLayerError() * SO3weight4;
        summation4 = SH4SO1Error + SH4SO2Error + SH4SO3Error;

        double SH5SO1Error = getH5OutputLayerError() * SO1weight5;
        double SH5SO2Error = getH5OutputLayerError() * SO2weight5;
        double SH5SO3Error = getH5OutputLayerError() * SO3weight5;
        summation5 = SH5SO1Error + SH5SO2Error + SH5SO3Error;

        double SH6SO1Error = getH6OutputLayerError() * SO1weight6;
        double SH6SO2Error = getH6OutputLayerError() * SO2weight6;
        double SH6SO3Error = getH6OutputLayerError() * SO3weight6;
        summation6 = SH6SO1Error + SH6SO2Error + SH6SO3Error;
    }

    @GetMapping("/train-ann")
    void trainPerceptron(String trainDataUrl) {
        int trainingNum = 0;
        this.loadDataset("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTraining.data");
```

```
        String plantName;

        //for all available data in the testing file
        for (int i = 0; i < irisLines.size(); i++) {
            //individually set values from file
            //sepal length
            input[0] = Double.parseDouble(irisValues.get(i)[0]);
            //sepal width
            input[1] = Double.parseDouble(irisValues.get(i)[1]);
            //petal length
            input[2] = Double.parseDouble(irisValues.get(i)[2]);
            //petal width
            input[3] = Double.parseDouble(irisValues.get(i)[3]);
            //plant name
            plantName = irisValues.get(i)[4];

            System.out.println("-------- 5-6-3 ANN IRIS DATASET TRAINING " +
trainingNum + "-------");
            int epochNum = 0;

            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));

                //output current epoch results
                //calculate the weighted sums into the output layer
                //finally the output from the output layer
                setO1Output();
                setO2Output();
                setO3Output();
                //compare values
                this.comparePlants(input[0], input[1], input[2], input[3]);

                this.updateSOxWeights();

                //summation
                this.setSummations();

                this.updateSHxWeights();

                System.out.println("Final values are: SO1 " + getO1Output() +
" SO2 " + getO2Output() + " SO3 " + getO3Output());

                //output current epoch results
                this.classify(getO1Output(), getO2Output(), getO3Output());

                System.out.println("Observed output: " + observedOutput + " |
Category: " + this.getObservedOutputName());
                System.out.println("Target output: " + targetOutput + " |
Category: " + this.targetOutputName);
                System.out.println("input values are: " + input[0] + ", " +
input[1] + ", " + input[2] + ", " + input[3] + ", " + plantName + "\n");

                //increment epoch number
                epochNum++;
                //continue loop until values are equal
            } while (!Objects.equals(targetOutputName, observedOutputName) &&
```

```
!Objects.equals(plantName, observedOutputName));
            trainingNum++;
        }
    }

    //execute the perceptron
    @GetMapping("/test-ann")
    String executePerceptron(String testDataUrl) {
        int iterationNum = 0;
        this.loadDataset("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTesting.data");
        String plantName;
        correctlyClassified = 0;

        //for all available data in the testing file
        for (int i = 0; i < irisLines.size(); i++) {
            //individually set values from file
            //sepal length
            input[0] = Double.parseDouble(irisValues.get(i)[0]);
            //sepal width
            input[1] = Double.parseDouble(irisValues.get(i)[1]);
            //petal length
            input[2] = Double.parseDouble(irisValues.get(i)[2]);
            //petal width
            input[3] = Double.parseDouble(irisValues.get(i)[3]);
            //plant name
            plantName = irisValues.get(i)[4];

            System.out.println("--------- 5-6-3 ANN IRIS DATASET OFFICIAL RUN
" + iterationNum + "---------");
            //set epoch to zero for looping
            int epochNum = 0;

            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));

                //output current epoch results
                //calculate the weighted sums into the output layer
                //finally the output from the output layer
                setO1Output();
                setO2Output();
                setO3Output();

                //compare values
                this.comparePlants(input[0], input[1], input[2], input[3]);

                //step 3: calculate the error between actual output and
desired output
                //See outputLayerErrorCalculation function

                //step 4: adjust each weight in the connections between
output and hidden layers
                this.updateSOxWeights();

                //step 5: calculate error between input and hidden layer
(part of step 6's function)
```

```
                this.setSummations();

                //step 6: adjust each weight in the connections between
hidden and input layers
                this.updateSHxWeights();

                //output from output layer
                System.out.println("Final values are: SO1 " + getO1Output() +
" SO2 " + getO2Output() + " SO3 " + getO3Output());

                //classify the values
                classify(getO1Output(), getO2Output(), getO3Output());

                //output results
                System.out.println("Observed output: " + observedOutput + " |
Category: " + this.getObservedOutputName());
                System.out.println("Target output: " + targetOutput + " |
Category: " + this.targetOutputName);
                System.out.println("input values are: " + input[0] + ", " +
input[1] + ", " + input[2] + ", " + input[3] + ", " + plantName + "\n");

            /* System.out.println("Setosa points " + setosaPoints);
             System.out.println("Versicolour points " + versicolourPoints);
             System.out.println("Virginica points " + virginicaPoints);*/
                //increment epoch number
                epochNum++;

                //Calculate Network Error
                double O1NetError = differenceSquared(setosaTargetOutput,
getO1Output());
                double O2NetError =
differenceSquared(versicolourTargetOutput, getO2Output());
                double O3NetError =
differenceSquared(versicolourTargetOutput, getO3Output());
                OutputLayerNetError = O1NetError + O2NetError + O3NetError;
                //continue loop until values are equal
            } while (!Objects.equals(targetOutputName, observedOutputName));

            //Calculate Network Accuracy
            if (plantName.equals(observedOutputName) &&
plantName.equals(targetOutputName)) {
                this.correctlyClassified++;
            }
            this.networkAccuracy = (correctlyClassified /
irisLines.size())*100;
            this.networkError += OutputLayerNetError;
            iterationNum++;
        }
        return "Accuracy of the 5-6-3 ANN is: " + networkAccuracy + "%";
    }

    void loadDataset(String url) {
        irisLines.clear();
        irisValues.clear();
        //receive input from file
        try {
            //File irisData = new File("/Users/boost/IdeaProjects/AI Test
```

```
2/src/iris/iris.data");

            File irisData = new File(url);
            Scanner scanner = new Scanner(irisData);
            while (scanner.hasNext()) {
                String next = scanner.next();
                //prints the data for testing
                //System.out.println(next);
                irisLines.add(next);
            }

            scanner.close();
        } catch (Exception e) {
            System.out.println("error " + e);
        }

        //assign all values to a list of arrays for value separation
        for (int i = 0; i < irisLines.size(); i++) {
            irisValues.add(irisLines.get(i).split(","));
        }


        //test output
        //System.out.println("Display test " + irisLines.get(0) + " and " +
irisValues.get(0)[0]);

        //Iris Setosa Samples ( 0 - 50)
        for (int samples = 0; samples < irisLines.size() / 3; samples++) {
            //Sepal Length of sample
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            setosaSepalLengthAvg += sepalLength;

            //get maximum sepal length for the setosa
            if (setosaSepalLengthMax < sepalLength) {
                setosaSepalLengthMax = sepalLength;
            }

            //get minimum sepal length for the setosa
            if (setosaSepalLengthMin > sepalLength) {
                setosaSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            setosaSepalWidthAvg += sepalWidth;

            //get maximum sepal width for the setosa
            if (setosaSepalWidthMax < sepalWidth) {
                setosaSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the setosa
            if (setosaSepalWidthMin > sepalWidth) {
                setosaSepalWidthMin = sepalWidth;
            }
```

```
            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            setosaPetalLengthAvg += petalLength;

            //max petal length of setosa
            if (setosaPetalLengthMax < petalLength) {
                setosaPetalLengthMax = petalLength;
            }

            //min petal length of setosa
            if (setosaPetalLengthMin > petalLength) {
                setosaPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            setosaPetalWidthAvg += petalWidth;

            //max petal width of setosa
            if (setosaPetalWidthMax < petalWidth) {
                setosaPetalWidthMax = petalWidth;
            }

            //min petal width of setosa
            if (setosaPetalWidthMin > petalWidth) {
                setosaPetalWidthMin = petalWidth;
            }

            //Class Name of sample
            setosaName = irisValues.get(samples)[4];

        }
        setosaSepalLengthAvg = setosaSepalLengthAvg / (irisLines.size() / 3);
        setosaSepalWidthAvg = setosaSepalWidthAvg / (irisLines.size() / 3);
        setosaPetalLengthAvg = setosaPetalLengthAvg / (irisLines.size() / 3);
        setosaPetalWidthAvg = setosaPetalWidthAvg / (irisLines.size() / 3);
        setosaTotalAvg = (setosaPetalLengthAvg + setosaPetalWidthAvg +
setosaSepalLengthAvg + setosaSepalWidthAvg) / 4;

        System.out.println("                        Min   Max   Mean");
        System.out.println("setosa sepal length:  " + setosaSepalLengthMin +
"   " + setosaSepalLengthMax + "   " + setosaSepalLengthAvg);
        System.out.println("setosa sepal width:   " + setosaSepalWidthMin + "
" + setosaSepalWidthMax + "   " + setosaSepalWidthAvg);
        System.out.println("setosa petal length:  " + setosaPetalLengthMin +
"   " + setosaPetalLengthMax + "   " + setosaPetalLengthAvg);
        System.out.println("setosa petal width:   " + setosaPetalWidthMin + "
" + setosaPetalWidthMax + "   " + setosaPetalWidthAvg);
        System.out.println("setosa total average: " + setosaTotalAvg + "\n");

        //Iris Versicolour Samples (50 - 100)
        for (int samples = irisLines.size() / 3; samples < (irisLines.size()
/ 3) * 2; samples++) {
            //Sepal Length of sample
```

```
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            versicolourSepalLengthAvg += sepalLength;

            //get maximum sepal length for the versicolour
            if (versicolourSepalLengthMax < sepalLength) {
                versicolourSepalLengthMax = sepalLength;
            }

            //get minimum sepal length for the versicolour
            if (versicolourSepalLengthMin > sepalLength) {
                versicolourSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            versicolourSepalWidthAvg += sepalWidth;

            //get maximum sepal width for the versicolour
            if (versicolourSepalWidthMax < sepalWidth) {
                versicolourSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the versicolour
            if (versicolourSepalWidthMin > sepalWidth) {
                versicolourSepalWidthMin = sepalWidth;
            }

            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            versicolourPetalLengthAvg += petalLength;

            //max petal length of versicolour
            if (versicolourPetalLengthMax < petalLength) {
                versicolourPetalLengthMax = petalLength;
            }

            //min petal length of versicolour
            if (versicolourPetalLengthMin > petalLength) {
                versicolourPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            versicolourPetalWidthAvg += petalWidth;

            //max petal width of versicolour
            if (versicolourPetalWidthMax < petalWidth) {
                versicolourPetalWidthMax = petalWidth;
            }

            //min petal width of versicolour
            if (versicolourPetalWidthMin > petalWidth) {
                versicolourPetalWidthMin = petalWidth;
```

```
            }

            //Class Name of sample
            versicolourName = irisValues.get(samples)[4];

        }
        versicolourSepalLengthAvg = versicolourSepalLengthAvg /
(irisLines.size() / 3);
        versicolourSepalWidthAvg = versicolourSepalWidthAvg /
(irisLines.size() / 3);
        versicolourPetalLengthAvg = versicolourPetalLengthAvg /
(irisLines.size() / 3);
        versicolourPetalWidthAvg = versicolourPetalWidthAvg /
(irisLines.size() / 3);
        versicolourTotalAvg = (versicolourPetalLengthAvg +
versicolourPetalWidthAvg + versicolourSepalLengthAvg +
versicolourSepalWidthAvg) / 4;

        System.out.println("                              Min   Max   Mean");
        System.out.println("versicolour sepal length:  " +
versicolourSepalLengthMin + "   " + versicolourSepalLengthMax + "   " +
versicolourSepalLengthAvg);
        System.out.println("versicolour sepal width:   " +
versicolourSepalWidthMin + "   " + versicolourSepalWidthMax + "   " +
versicolourSepalWidthAvg);
        System.out.println("versicolour petal length:  " +
versicolourPetalLengthMin + "   " + versicolourPetalLengthMax + "   " +
versicolourPetalLengthAvg);
        System.out.println("versicolour petal width:   " +
versicolourPetalWidthMin + "   " + versicolourPetalWidthMax + "   " +
versicolourPetalWidthAvg);
        System.out.println("versicolour total average: " +
versicolourTotalAvg + "\n");

        //Iris virginica Samples (100 - 150)
        for (int samples = (irisLines.size() / 3) * 2; samples <
(irisLines.size() / 3) * 3; samples++) {
            //Sepal Length of sample
            double sepalLength =
Double.parseDouble(irisValues.get(samples)[0]);
            virginicaSepalLengthAvg += sepalLength;

            //get maximum sepal length for the virginica
            if (virginicaSepalLengthMax < sepalLength) {
                virginicaSepalLengthMax = sepalLength;
            }

            //get minimum sepal length for the virginica
            if (virginicaSepalLengthMin > sepalLength) {
                virginicaSepalLengthMin = sepalLength;
            }

            //Sepal Width of sample
            double sepalWidth =
Double.parseDouble(irisValues.get(samples)[1]);
            virginicaSepalWidthAvg += sepalWidth;
```

```
            //get maximum sepal width for the virginica
            if (virginicaSepalWidthMax < sepalWidth) {
                virginicaSepalWidthMax = sepalWidth;
            }

            //get minimum sepal width for the virginica
            if (virginicaSepalWidthMin > sepalWidth) {
                virginicaSepalWidthMin = sepalWidth;
            }

            //Petal Length of sample
            double petalLength =
Double.parseDouble(irisValues.get(samples)[2]);
            virginicaPetalLengthAvg += petalLength;

            //max petal length of virginica
            if (virginicaPetalLengthMax < petalLength) {
                virginicaPetalLengthMax = petalLength;
            }

            //min petal length of virginica
            if (virginicaPetalLengthMin > petalLength) {
                virginicaPetalLengthMin = petalLength;
            }

            //Petal Width of sample
            double petalWidth =
Double.parseDouble(irisValues.get(samples)[3]);
            virginicaPetalWidthAvg += petalWidth;

            //max petal width of virginica
            if (virginicaPetalWidthMax < petalWidth) {
                virginicaPetalWidthMax = petalWidth;
            }

            //min petal width of virginica
            if (virginicaPetalWidthMin > petalWidth) {
                virginicaPetalWidthMin = petalWidth;
            }

            //Class Name of sample
            virginicaName = irisValues.get(samples)[4];


        }
        virginicaSepalLengthAvg = virginicaSepalLengthAvg / (irisLines.size()
/ 3);
        virginicaSepalWidthAvg = virginicaSepalWidthAvg / (irisLines.size() /
3);
        virginicaPetalLengthAvg = virginicaPetalLengthAvg / (irisLines.size()
/ 3);
        virginicaPetalWidthAvg = virginicaPetalWidthAvg / (irisLines.size() /
3);
        virginicaTotalAvg = (virginicaPetalLengthAvg + virginicaPetalWidthAvg
+ virginicaSepalLengthAvg + virginicaSepalWidthAvg) / 4;

        System.out.println("                          Min   Max   Mean");
```

```
        System.out.println("virginica sepal length:  " +
virginicaSepalLengthMin + "   " + virginicaSepalLengthMax + "   " +
virginicaSepalLengthAvg);
        System.out.println("virginica sepal width:   " +
virginicaSepalWidthMin + "   " + virginicaSepalWidthMax + "   " +
virginicaSepalWidthAvg);
        System.out.println("virginica petal length:  " +
virginicaPetalLengthMin + "   " + virginicaPetalLengthMax + "   " +
virginicaPetalLengthAvg);
        System.out.println("virginica petal width:   " +
virginicaPetalWidthMin + "   " + virginicaPetalWidthMax + "   " +
virginicaPetalWidthAvg);
        System.out.println("virginica total average: " + virginicaTotalAvg +
"\n");
    }

    double differenceSquared(double plantTargetOut, double
outputLayerNeuronResult) {
        double sumOutputError = Math.pow((plantTargetOut -
outputLayerNeuronResult), 2);
        return sumOutputError;
    }


    public static void main(String[] args) {
        //create necessary values
        Perceptron563 myPerceptron2 = new Perceptron563();

        //train the ANN
        //myPerceptron2.trainPerceptron("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTraining.data");

        //accept values manually for input (for testing & debugging)
        //myPerceptron2.inputPerceptron();

        //execute the ANN program
        /*myPerceptron2.executePerceptron("/Users/boost/IdeaProjects/AI Test
2/src/iris/irisTesting.data");
        System.out.println("Network Error: " + (0.5 *
myPerceptron2.networkError));
        System.out.println("Correctly Classified: " +
myPerceptron2.correctlyClassified);
        System.out.println("Network Accuracy: " +
myPerceptron2.networkAccuracy + "%");*/
    }
}
```

III: Any comments about this ANN based on your experience.

Building these ANN's have been a challenging yet pleasurable experience. I realize that the nature of building an ANN is like back-propagation in and of itself. The first "iteration" or session of my attempt at building the 5-2-3 ANN was almost like a blind journey or the initial "forward pass". I did not know the exact direction I was going but I knew the task that I wanted to achieve. After finding a lead, the next iteration of my understanding began. A few "iterations" or sessions of effort later, I realized that I was using back-propagation in real life to achieve the goal of completing the ANN. My code and my calculations were inaccurate in the beginning but I continued to sharpen my understanding of the concepts and gain familiarity with the desired outcomes and calculations. Though I found some of the formula notations hard to grasp at times, I was able to cross-reference provided examples to clarify my understanding until I reached the desired outcome. This was a fun Test. I spent a total of over 27 active working hours on the completion of this Test.