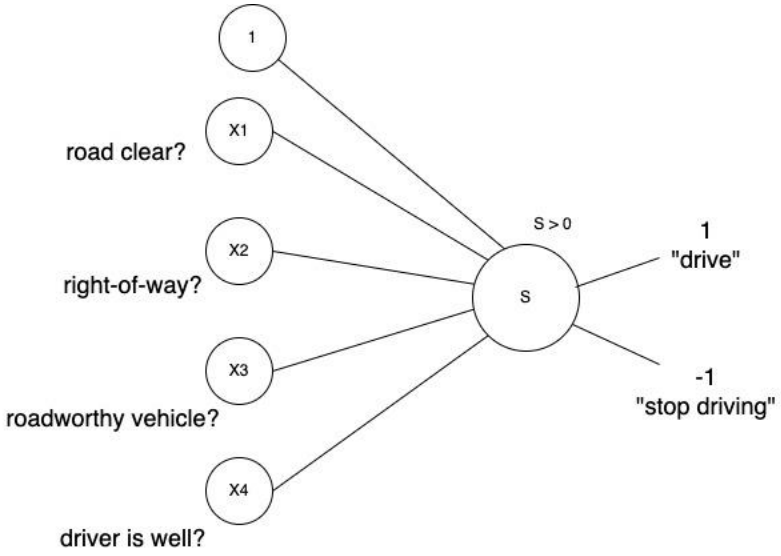
I: My information.

// Course: CS3642-01
// Student name: Andujar Brutus
// Student ID: 0001002015
// Assignment #: 3
// Due Date: 11/20/2023
// Signature: Andujar Brutus (Your signature assures that everything is your own work. Required)
// Score: _____(Note: Score will be posted on D2L)

II: My perception architecture for a real-world application.

My custom perceptron is designed for the real-world application of driving a vehicle. From any point on the route from start to finish, any of the input neurons may vary. This perceptron is made to mitigate risk and possibly save lives.



III: My Source Code:

Task 1 Source Code:

```
package com.example.aiprog3;

import org.springframework.web.bind.annotation.*;

import java.util.*;
import java.lang.*;
import java.util.Random;

@CrossOrigin(origins = "http://localhost:3000/")
@RestController
@RequestMapping(value = "/perceptron")
public class Perceptron {
    //scanner to receive input
    private final Scanner scanner = new Scanner(System.in);

    //array to store input positions
    private final int[][] image = new int[2][2];

    //store variable information
    private double targetOutput, observedOutput;
    private final int BIAS_NEURON = 1;
    private int x1;
    private int x2;
    private int x3;
    private int x4;
    private String observedOutputColor;
    private String targetOutputColor;

    //initialize weights with random values between [-0.5 and 0.5]
    private double weight0 = ((Math.random()) - 0.5);
    private double weight1 = ((Math.random()) - 0.5);
    private double weight2 = ((Math.random()) - 0.5);
    private double weight3 = ((Math.random()) - 0.5);
    private double weight4 = ((Math.random()) - 0.5);
    private int darkNum, brightNum;

    //default constructor
    Perceptron() {
    }

    //get output color string
    String getOutputColor() {
        return this.observedOutputColor;
    }

    //set output color string
    void setOutputColor(String newOutputColor) {
        this.observedOutputColor = newOutputColor;
    }

    @GetMapping("/input/x1")
    public int getX1() {
        return x1;
```

```
    }

    @PostMapping("/input/x1/{x1}")
    public void setX1(@PathVariable int x1) {
        this.x1 = x1;
    }

    @GetMapping("/input/x2")
    public int getX2() {
        return x2;
    }

    @PostMapping("/input/x2/{x2}")
    public void setX2(@PathVariable int x2) {
        this.x2 = x2;
    }

    @GetMapping("/input/x3")
    public int getX3() {
        return x3;
    }

    @PostMapping("/input/x3/{x3}")
    public void setX3(@PathVariable int x3) {
        this.x3 = x3;
    }

    @GetMapping("/input/x4")
    public int getX4() {
        return x4;
    }

    @PostMapping("/input/x4/{x4}")
    public void setX4(@PathVariable int x4) {
        this.x4 = x4;
    }

    @GetMapping("/train-perceptron")
    void trainPerceptron() {
        int classified;
        int trainingNum = 0;
        Random random = new Random();

        //for half of all available patterns (16 / 2 = 8)
        for (int i = 0; i < 8; i++) {
            //individually set 1 or -1 for each image space
            for (int j = 0; j < 2; j++) {
                image[0][0] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                image[1][0] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                image[0][1] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                image[1][1] = (random.nextBoolean()) ? 1 : -1;
```

```
            }

            //then compare as normal
            this.compareBrightDark();
            System.out.println("--------BRIGHT/DARK PERCEPTRON TRAINING " +
trainingNum + "-------");
            int epochNum = 0;
            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));
                this.displayWeightData();
                //output current epoch results
                classified = this.classify(this.S());
                System.out.println("Observed output: " + classified + " |
Category: " + this.getOutputColor());
                System.out.println("Target output: " + targetOutput + " |
Category: " + targetOutputColor);
                System.out.println("X values are: " + image[0][0] + ", " +
image[0][1] + ", " + image[1][0] + ", " + image[1][1] + "\n");

                //update weights & learning rate if necessary
                this.updateWeights(targetOutput, observedOutput);

                //increment epoch number
                epochNum++;
                //continue loop until values are equal
            } while (targetOutput != observedOutput);
            trainingNum++;
        }
    }

    @PostMapping("/input-perceptron")
        //initialize x values
    String inputPerceptron() {
        //test input
       /* System.out.println("enter 1 for bright & -1 for dark");
        System.out.println("Enter value for pixel 1 (top left)");
        setX1(scanner.nextInt());
        System.out.println("Enter value for pixel 2 (top right)");
        setX2(scanner.nextInt());
        System.out.println("Enter value for pixel 3 (bottom left)");
        setX3(scanner.nextInt());
        System.out.println("Enter value for pixel 4 (bottom right)");
        setX4(scanner.nextInt());*/

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

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

```
        //compare tile colors
        this.compareBrightDark();
        return "The input values are " + x1 + ", " + x2 + ", " + x3 + ", " +
x4 + ".";
    }

    //compare the bright vs dark tiles
    void compareBrightDark() {
        brightNum = 0;
        darkNum = 0;

        //count number of bright vs dark pixels in the image array
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                if (image[i][j] == 1) {
                    this.brightNum++;
                } else {
                    this.darkNum++;
                }
            }
        }

        //calculate the target output using bright vs dark pixel count
        if (brightNum >= 2) {
            targetOutputColor = "bright";
            targetOutput = 1;
        } else {
            targetOutputColor = "dark";
            targetOutput = -1;
        }
    }


    //classify the image
    int classify(double SCalculation) {
        if (SCalculation > 0) {
            //return as bright
            this.setOutputColor("bright");
            this.observedOutput = 1;
            return 1;
        } else {
            //otherwise return as dark
            this.setOutputColor("dark");
            this.observedOutput = -1;
            return -1;
        }
    }

    //Output layer calculation
    double S() {
        return (BIAS_NEURON * this.weight0) + ((this.x1 * this.weight1) +
(this.x2 * this.weight2) + (this.x3 * this.weight3) + (this.x4 *
this.weight4));
    }

    //Update weights if necessary
    void updateWeights(double targetOutput, double observedOutput) {
```

```
        if (targetOutput != observedOutput) {
            this.weight0 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), BIAS_NEURON);
            this.weight1 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x1);
            this.weight2 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x2);
            this.weight3 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x3);
            this.weight4 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x4);
        }
    }

    //displays weight values
    void displayWeightData() {
        System.out.println("weight values: " + this.weight0 + " " +
this.weight1 + " " + this.weight2 + " " + this.weight3 + " " + this.weight4);
    }

    //calculate the learning rate
    double learningRate(double targetOutput, double observedOutput) {
        return (targetOutput - observedOutput) * 0.1;
    }

    //calculate DeltaI
    double DeltaI(double learningRate, int pixelValue) {
        return learningRate * pixelValue;
    }

    @GetMapping("/execute-perceptron")
        //execute the perceptron
    String executePerceptron() {
        System.out.println("---------BRIGHT/DARK PERCEPTRON OFFICIAL RUN-----
----");
        int classified;
        //set epoch to zero for looping
        int epochNum = 0;

        do {
            //show epoch and weight information
            System.out.println("epoch " + (epochNum));
            this.displayWeightData();
            //output current epoch results
            classified = this.classify(this.S());
            System.out.println("Observed output: " + classified + " |
Category: " + this.getOutputColor());
            System.out.println("Target output: " + targetOutput + " |
Category: " + targetOutputColor);
            System.out.println("X values are: " + getX1() + ", " + getX2() +
", " + getX3() + ", " + getX4() + "\n");

            //update weights & learning rate if necessary
            this.updateWeights(targetOutput, observedOutput);

            //increment epoch number
            epochNum++;
```

```
            //continue loop until values are equal
        } while (targetOutput != observedOutput);

        return "\nepoch " + (epochNum - 1) + "\nObserved output: " +
classified + " | Category: " + this.getOutputColor() + "\nTarget output: " +
targetOutput + " | Category: " + targetOutputColor;
    }

    public static void main(String[] args) {
        //create necessary values
        //Perceptron myPerceptron = new Perceptron();
        //TESTING VALUES
        //accept values for each pixel
        //myPerceptron.inputPerceptron();
        //execute the perceptron program
        //myPerceptron.executePerceptron();
        //myPerceptron.trainPerceptron();
    }
}
```

Task 2 Source Code:

```
package com.example.aiprog3;

import org.springframework.web.bind.annotation.*;

import java.util.*;
import java.lang.*;
import java.util.Random;

@CrossOrigin(origins = "http://localhost:3000/")
@RestController
@RequestMapping(value = "/perceptron/driving")
public class DrivingPerceptron {
    //scanner to receive input
    private final Scanner scanner = new Scanner(System.in);

    //array to store input positions
    private final int[] conditions = new int[4];

    //store variable information
    private double targetOutput, observedOutput;
    private final int BIAS_NEURON = 1;
    private int x1;
    private int x2;
    private int x3;
    private int x4;
    private String observedOutputAction;
    private String targetOutputAction;

    //initialize weights with random values between [-0.5 and 0.5]
    private double weight0 = ((Math.random()) - 0.5);
    private double weight1 = ((Math.random()) - 0.5);
    private double weight2 = ((Math.random()) - 0.5);
    private double weight3 = ((Math.random()) - 0.5);
```

```
    private double weight4 = ((Math.random()) - 0.5);
    private int stopDriveNum, driveNum;

    //default constructor
    DrivingPerceptron() {
    }

    //get output color string
    String getOutputAction() {
        return this.observedOutputAction;
    }

    //set output color string
    void setOutputAction(String newOutputAction) {
        this.observedOutputAction = newOutputAction;
    }

    @GetMapping("/input/x1")
    public int getX1() {
        return x1;
    }

    @PostMapping("/input/x1/{x1}")
    public void setX1(@PathVariable int x1) {
        this.x1 = x1;
    }

    @GetMapping("/input/x2")
    public int getX2() {
        return x2;
    }

    @PostMapping("/input/x2/{x2}")
    public void setX2(@PathVariable int x2) {
        this.x2 = x2;
    }

    @GetMapping("/input/x3")
    public int getX3() {
        return x3;
    }

    @PostMapping("/input/x3/{x3}")
    public void setX3(@PathVariable int x3) {
        this.x3 = x3;
    }

    @GetMapping("/input/x4")
    public int getX4() {
        return x4;
    }

    @PostMapping("/input/x4/{x4}")
    public void setX4(@PathVariable int x4) {
        this.x4 = x4;
    }
```

```
    @GetMapping("/train-perceptron")
    void trainPerceptron() {
        int classified;
        int trainingNum = 0;
        Random random = new Random();

        //for half of all available patterns (16 / 2 = 8)
        for (int i = 0; i < 8; i++) {
            //individually set 1 or -1 for each condition space
            for (int j = 0; j < 2; j++) {
                conditions[0] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                conditions[1] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                conditions[2] = (random.nextBoolean()) ? 1 : -1;
            }
            for (int j = 0; j < 2; j++) {
                conditions[3] = (random.nextBoolean()) ? 1 : -1;
            }

            //then compare as normal
            this.compareDriveStopDrive();
            System.out.println("--------DRIVING PERCEPTRON TRAINING " +
trainingNum + "-------");
            int epochNum = 0;
            do {
                //show epoch and weight information
                System.out.println("epoch " + (epochNum));
                this.displayWeightData();
                //output current epoch results
                classified = this.classify(this.S());
                System.out.println("Observed output: " + classified + " |
Category: " + this.getOutputAction());
                System.out.println("Target output: " + targetOutput + " |
Category: " + targetOutputAction);
                System.out.println("X values are: " + conditions[0] + ", " +
conditions[1] + ", " + conditions[2] + ", " + conditions[3] + "\n");

                //update weights & learning rate if necessary
                this.updateWeights(targetOutput, observedOutput);

                //increment epoch number
                epochNum++;
                //continue loop until values are equal
            } while (targetOutput != observedOutput);
            trainingNum++;
        }
    }

    @PostMapping("/input-perceptron")
        //initialize x values
    String inputPerceptron() {
        //test input
       /* System.out.println("enter 1 for yes & -1 for no");
        System.out.println("Is the road clear?");
```

```
        setX1(scanner.nextInt());
        System.out.println("Do you have the right-of-way?");
        setX2(scanner.nextInt());
        System.out.println("Is the car roadworthy?");
        setX3(scanner.nextInt());
        System.out.println("Is the driver well?");
        setX4(scanner.nextInt());*/

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

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

        //compare tile colors
        this.compareDriveStopDrive();
        return "The input values are " + x1 + ", " + x2 + ", " + x3 + ", " +
x4 + ".";
    }

    //compare the bright vs dark tiles
    void compareDriveStopDrive() {
        stopDriveNum = 0;
        driveNum = 0;

        //count number of drive vs stop conditions in the conditions array
        for (int i = 0; i < 4; i++) {
                if (conditions[i] == 1) {
                    this.driveNum++;
                } else {
                    this.stopDriveNum++;
                }
        }

        //calculate the target output using bright vs dark pixel count
        if (driveNum == 4) {
            targetOutputAction = "drive";
            targetOutput = 1;
        } else {
            targetOutputAction = "stop driving";
            targetOutput = -1;
        }
    }


    //classify the image
    int classify(double SCalculation) {
        if (SCalculation > 0) {
            //return as bright
            this.setOutputAction("drive");
            this.observedOutput = 1;
```

```
            return 1;
        } else {
            //otherwise return as dark
            this.setOutputAction("stop driving");
            this.observedOutput = -1;
            return -1;
        }
    }

    //Output layer calculation
    double S() {
        return (BIAS_NEURON * this.weight0) + ((this.x1 * this.weight1) +
(this.x2 * this.weight2) + (this.x3 * this.weight3) + (this.x4 *
this.weight4));
    }

    //Update weights if necessary
    void updateWeights(double targetOutput, double observedOutput) {
        if (targetOutput != observedOutput) {
            this.weight0 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), BIAS_NEURON);
            this.weight1 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x1);
            this.weight2 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x2);
            this.weight3 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x3);
            this.weight4 += this.DeltaI(this.learningRate(targetOutput,
observedOutput), this.x4);
        }
    }

    //displays weight values
    void displayWeightData() {
        System.out.println("weight values: " + this.weight0 + " " +
this.weight1 + " " + this.weight2 + " " + this.weight3 + " " + this.weight4);
    }

    //calculate the learning rate
    double learningRate(double targetOutput, double observedOutput) {
        return (targetOutput - observedOutput) * 0.1;
    }

    //calculate DeltaI
    double DeltaI(double learningRate, int driveValue) {
        return learningRate * driveValue;
    }

    @GetMapping("/execute-perceptron")
        //execute the perceptron
    String executePerceptron() {
        System.out.println("---------DRIVING PERCEPTRON OFFICIAL RUN---------
");
        int classified;
        //set epoch to zero for looping
        int epochNum = 0;
```

```
        do {
            //show epoch and weight information
            System.out.println("epoch " + (epochNum));
            this.displayWeightData();
            //output current epoch results
            classified = this.classify(this.S());
            System.out.println("Observed output: " + classified + " |
Category: " + this.getOutputAction());
            System.out.println("Target output: " + targetOutput + " |
Category: " + targetOutputAction);
            System.out.println("X values are: " + getX1() + ", " + getX2() +
", " + getX3() + ", " + getX4() + "\n");

            //update weights & learning rate if necessary
            this.updateWeights(targetOutput, observedOutput);

            //increment epoch number
            epochNum++;
            //continue loop until values are equal
        } while (targetOutput != observedOutput);

        return "\nepoch " + (epochNum - 1) + "\nObserved output: " +
classified + " | Category: " + this.getOutputAction() + "\nTarget output: " +
targetOutput + " | Category: " + targetOutputAction;
    }

    public static void main(String[] args) {
        //create necessary values
        //DrivingPerceptron drivePerceptron = new DrivingPerceptron();
        //TESTING VALUES
        //accept values for each pixel
        //myPerceptron.inputPerceptron();
        //execute the perceptron program
        //myPerceptron.executePerceptron();
        //drivePerceptron.trainPerceptron();
    }
}
```