Machine Learning Program

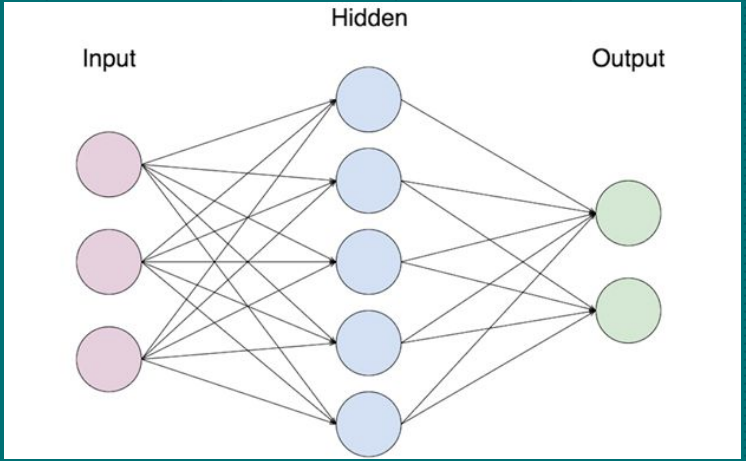
By Srinath Rangan

1.
What Does it Do?

Program Purpose

Program Summary

- Set document of 250 given inputs
- Set document of known outputs to given inputs in binary
- Makes network that generates a formula that matches most inputs with given output



2.
How It Functions

What makes up a neural network?

Building Block Classes

Calculate

Given an input value, uses a formula to calculate an output based on bias and weights.

Hidden Neurons

In charge of weights and bias, initializes them to a random value.

Backward Propagation

Goes back a fixes the formula to accommodate for a incorrectly matched input and output..

Sigmoid Function

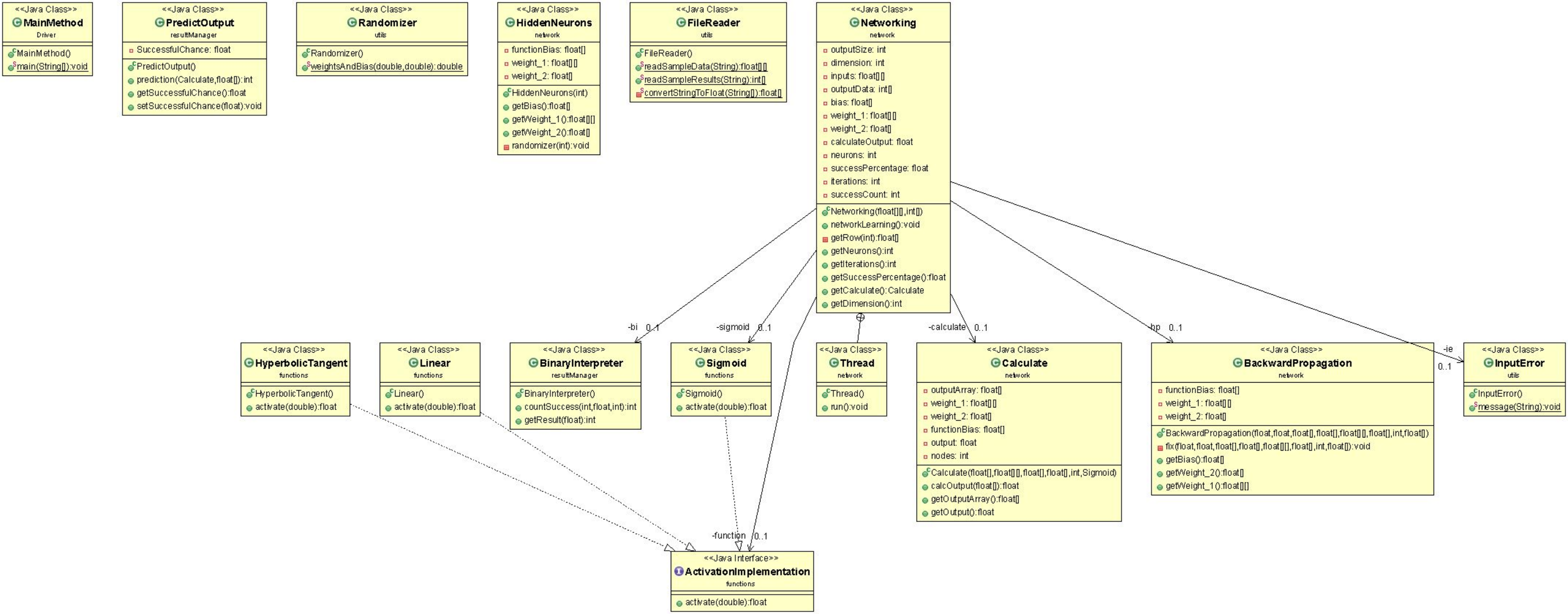
Calculates a value in between one and zero that is used by the Calculate class.

Binary Interpreter

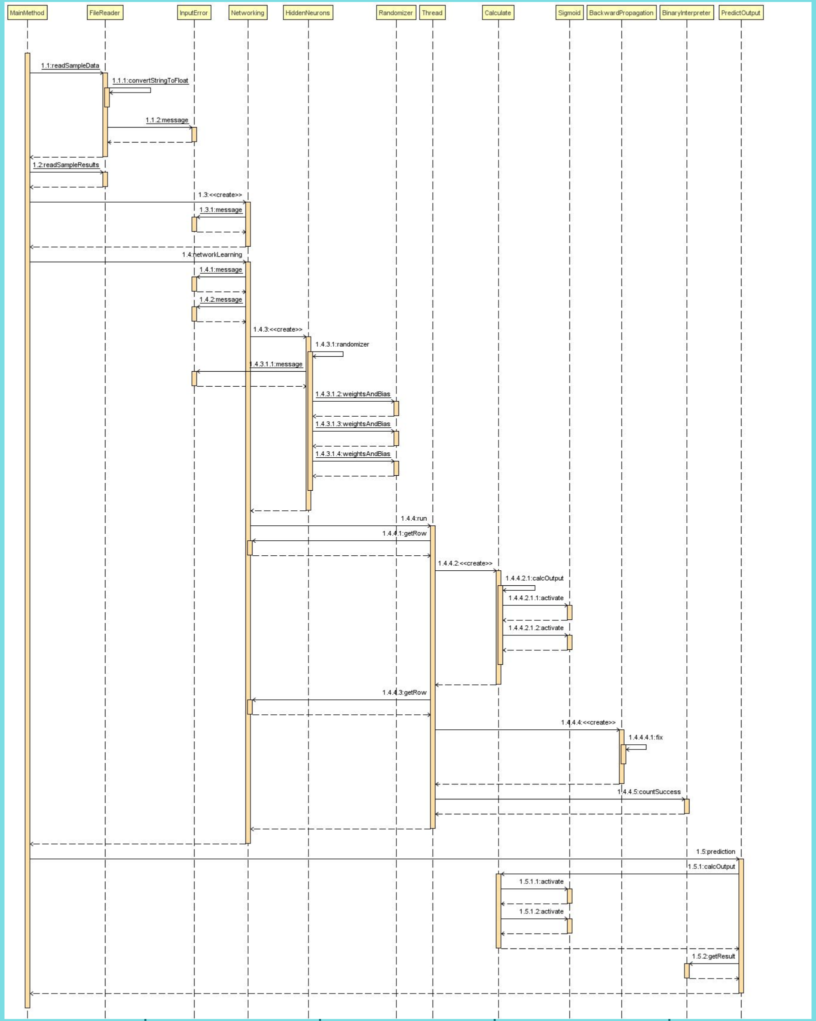
Determines the user’s input’s result and also counts how many iterations were successful.

Networking

Contains all the information about the network. Like the glue of the project.

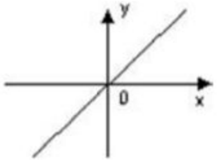
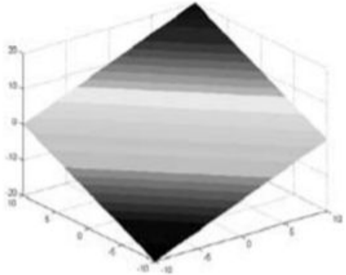
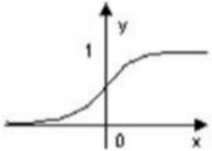
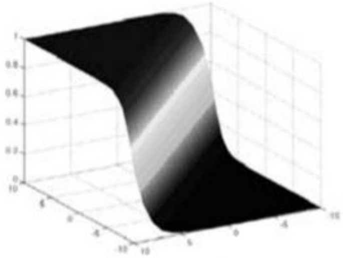
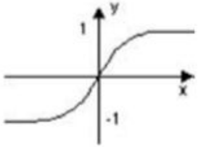
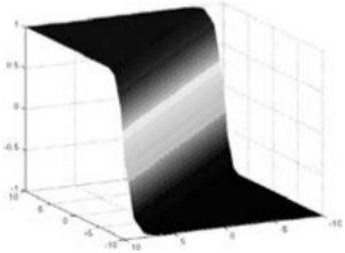


```
<<Java Class>>
MainMethod
Driver
MainMethod()
main(String[]):void

<<Java Class>>
PredictOutput
resultManager
SuccessfulChance: float
PredictOutput()
prediction(Calculate,float[]):int
getSuccessfulChance():float
setSuccessfulChance(float):void

<<Java Class>>
Randomizer
utils
Randomizer()
weightsAndBias(double,double):double

<<Java Class>>
HiddenNeurons
network
functionBias: float[]
weight_1: float[][]
weight_2: float[]
HiddenNeurons(int)
getBias():float[]
getWeight_1():float[][]
getWeight_2():float[]
randomizer(int):void

<<Java Class>>
FileReader
utils
FileReader()
readSampleData(String):float[][]
readSampleResults(String):int[]
convertStringToFloat(String[]):float[]

<<Java Class>>
Networking
network
outputSize: int
dimension: int
inputs: float[][]
outputData: int[]
bias: float[]
weight_1: float[][]
weight_2: float[]
calculateOutput: float
neurons: int
successPercentage: float
iterations: int
successCount: int
Networking(float[][],int[])
networkLearning():void
getRow(int):float[]
getNeurons():int
getIterations():int
getSuccessPercentage():float
getCalculate():Calculate
getDimension():int

<<Java Class>>
HyperbolicTangent
functions
HyperbolicTangent()
activate(double):float

<<Java Class>>
Linear
functions
Linear()
activate(double):float

<<Java Class>>
BinaryInterpreter
resultManager
BinaryInterpreter()
countSuccess(int,float,int):int
getResult(float):int

<<Java Class>>
Sigmoid
functions
Sigmoid()
activate(double):float

<<Java Class>>
Thread
network
Thread()
run():void

<<Java Class>>
Calculate
network
outputArray: float[]
weight_1: float[][]
weight_2: float[]
functionBias: float[]
output: float
nodes: int
Calculate(float[],float[][],float[],float[],int,Sigmoid)
calcOutput(float[]):float
getOutputArray():float[]
getOutput():float

<<Java Class>>
BackwardPropagation
network
functionBias: float[]
weight_1: float[][]
weight_2: float[]
BackwardPropagation(float,float,float[],float[],float[][],float[],int,float[])
fix(float,float,float[],float[],float[][],float[],int,float[]):void
getBias():float[]
getWeight_2():float[]
getWeight_1():float[][]

<<Java Class>>
InputError
utils
InputError()
message(String):void

<<Java Interface>>
ActivationImplementation
functions
activate(double):float
```


MainMethod
FileReader
InputError
Networking
HiddenNeurons
Randomizer
Thread
Calculate
Sigmoid
BackwardPropagation
BinaryInterpreter
PredictOutput
1.1:readSampleData
1.1.1:convertStringToFloat
1.1.2:message
1.2:readSampleResults
1.3:<<create>>
1.3.1:message
1.4:networkLearning
1.4.1:message
1.4.2:message
1.4.3:<<create>>
1.4.3.1:randomizer
1.4.3.1.1:message
1.4.3.1.2:weightsAndBias
1.4.3.1.3:weightsAndBias
1.4.3.1.4:weightsAndBias
1.4.4:run
1.4.4.1:getRow
1.4.4.2:<<create>>
1.4.4.2.1:calcOutput
1.4.4.2.1.1:activate
1.4.4.2.1.2:activate
1.4.4.3:getRow
1.4.4.4:<<create>>
1.4.4.4.1:fix
1.4.4.5:countSuccess
1.5:prediction
1.5.1:calcOutput
1.5.1.1:activate
1.5.1.2:activate
1.5.2:getResult

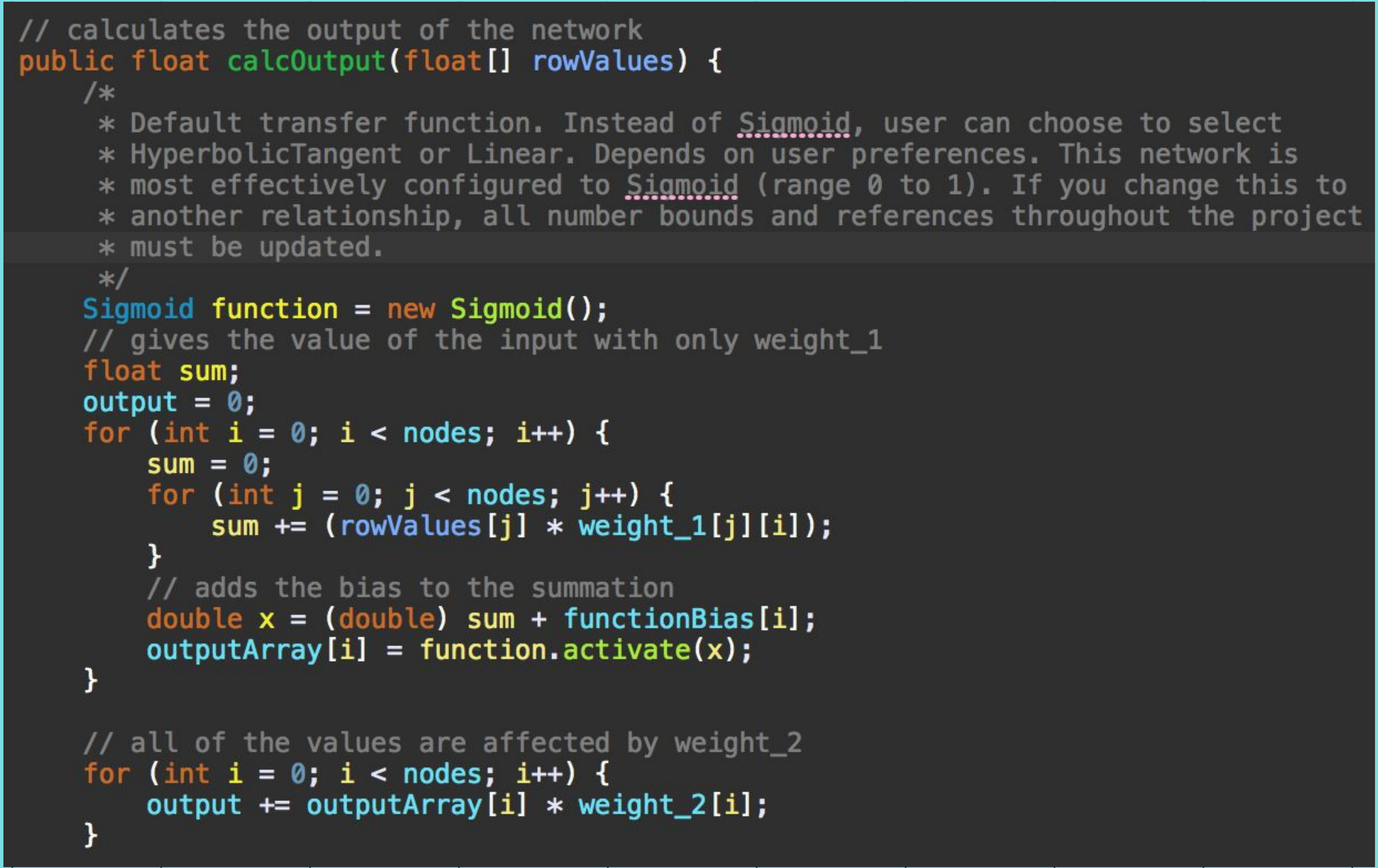
Sigmoid Class

- Always outputs a value in between 0 and 1
- If calculated value less than 0.5, then it’s 0
- If calculated value more than 0.5, then it’s 1

Activation Function	Mathematical Equation	2D Graphical Representation	3D Graphical Representation
Linear	$y = x$		
Sigmoid (logistic)	$y = \dfrac{1}{1 + e^{-x}}$		
Hyperbolic tangent	$y = \dfrac{1 - e^{-2x}}{1 + e^{2x}}$		

Calculate Class

- Uses sigmoid to calculate value
- Multiplies by weights
- Adds bias at the end
- Sum of two inputs in the row



```java
// calculates the output of the network
public float calcOutput(float[] rowValues) {
    /*
     * Default transfer function. Instead of Sigmoid, user can choose to select
     * HyperbolicTangent or Linear. Depends on user preferences. This network is
     * most effectively configured to Sigmoid (range 0 to 1). If you change this to
     * another relationship, all number bounds and references throughout the project
     * must be updated.
     */
    Sigmoid function = new Sigmoid();
    // gives the value of the input with only weight_1
    float sum;
    output = 0;
    for (int i = 0; i < nodes; i++) {
        sum = 0;
        for (int j = 0; j < nodes; j++) {
            sum += (rowValues[j] * weight_1[j][i]);
        }
        // adds the bias to the summation
        double x = (double) sum + functionBias[i];
        outputArray[i] = function.activate(x);
    }

    // all of the values are affected by weight_2
    for (int i = 0; i < nodes; i++) {
        output += outputArray[i] * weight_2[i];
    }
```

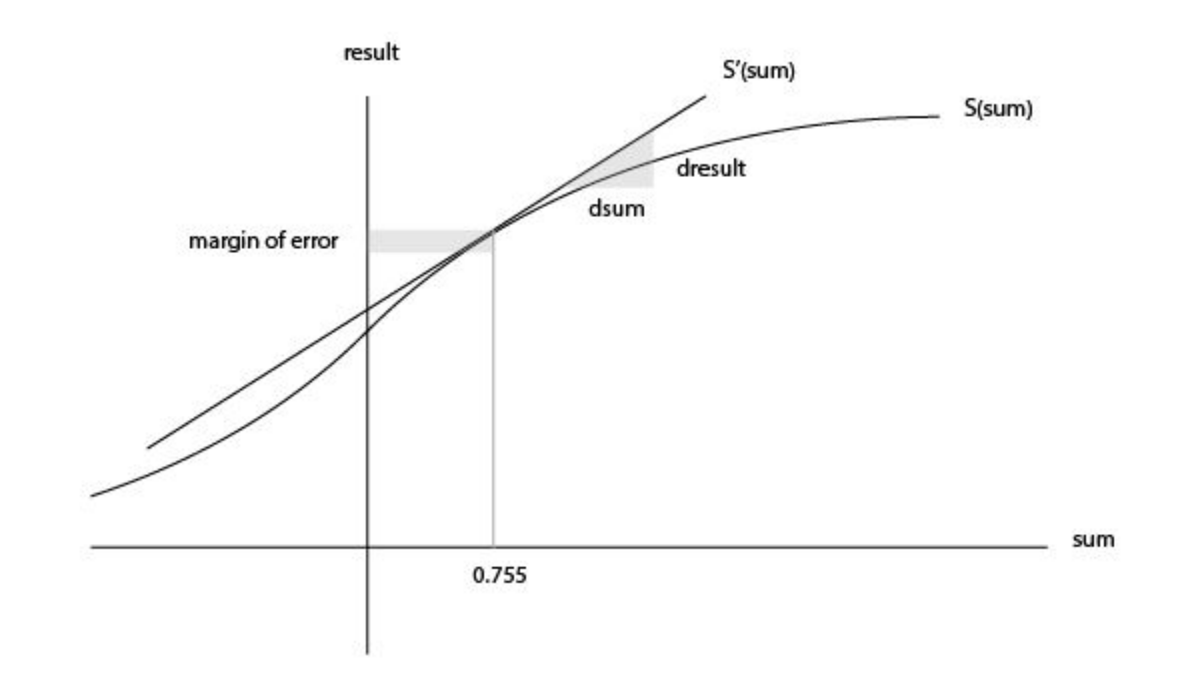
Fixing the Formula

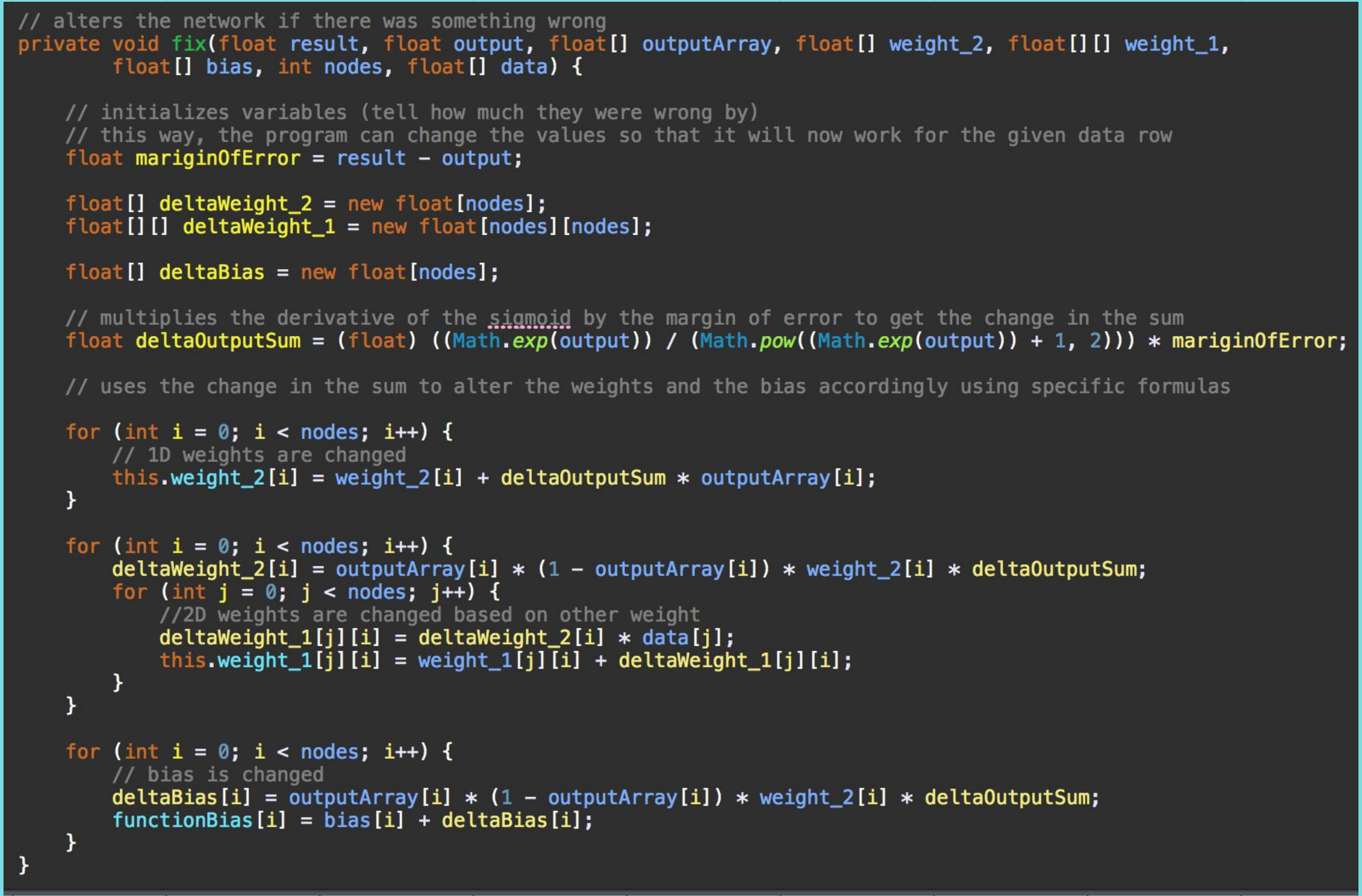
- Matches input with known result to calculate the margin of error
- Uses derivative of sigmoid to find correction factor with specific formulas
    - Fixes weights and bias

$$S(sum) = result$$

$$S'(sum) = \frac{dsum}{dresult}$$

$$\frac{\text{dsum}}{\text{dresult}} \times (\text{target result} - \text{calculated result}) = \triangle\text{sum}$$





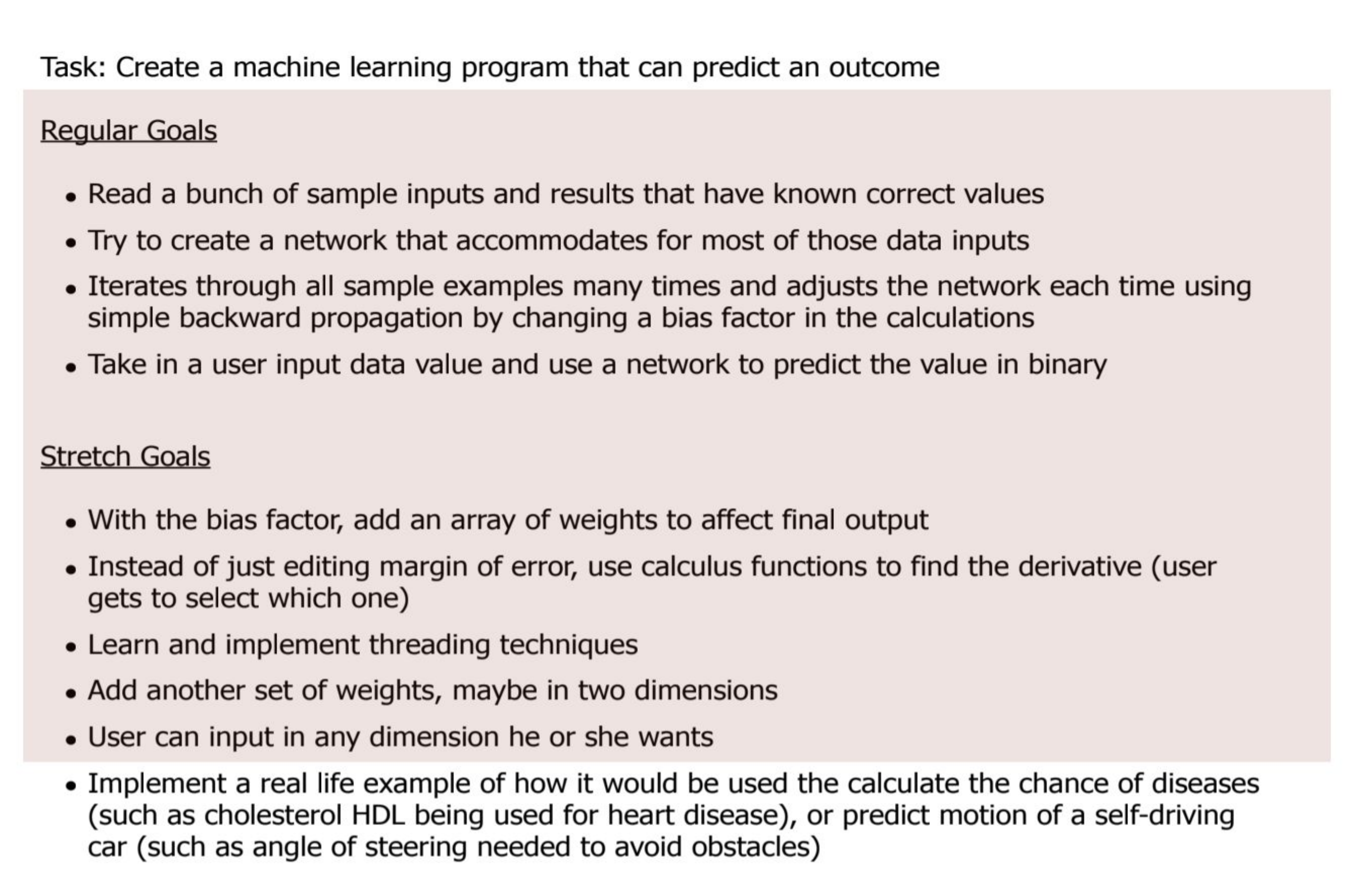
```
// alters the network if there was something wrong
private void fix(float result, float output, float[] outputArray, float[] weight_2, float[][] weight_1,
        float[] bias, int nodes, float[] data) {

    // initializes variables (tell how much they were wrong by)
    // this way, the program can change the values so that it will now work for the given data row
    float mariginOfError = result - output;

    float[] deltaWeight_2 = new float[nodes];
    float[][] deltaWeight_1 = new float[nodes][nodes];

    float[] deltaBias = new float[nodes];

    // multiplies the derivative of the sigmoid by the margin of error to get the change in the sum
    float deltaOutputSum = (float) ((Math.exp(output)) / (Math.pow((Math.exp(output)) + 1, 2))) * mariginOfError;

    // uses the change in the sum to alter the weights and the bias accordingly using specific formulas

    for (int i = 0; i < nodes; i++) {
        // 1D weights are changed
        this.weight_2[i] = weight_2[i] + deltaOutputSum * outputArray[i];
    }

    for (int i = 0; i < nodes; i++) {
        deltaWeight_2[i] = outputArray[i] * (1 - outputArray[i]) * weight_2[i] * deltaOutputSum;
        for (int j = 0; j < nodes; j++) {
            //2D weights are changed based on other weight
            deltaWeight_1[j][i] = deltaWeight_2[i] * data[j];
            this.weight_1[j][i] = weight_1[j][i] + deltaWeight_1[j][i];
        }
    }

    for (int i = 0; i < nodes; i++) {
        // bias is changed
        deltaBias[i] = outputArray[i] * (1 - outputArray[i]) * weight_2[i] * deltaOutputSum;
        functionBias[i] = bias[i] + deltaBias[i];
    }
}
```

Iterations

- Goes through 250 inputs 12,345 times
  - Total = 250 * 12,345 =3,086,520 times
- In final iteration:
  - Counts how many times the network successfully predicts the outcome
  - After 250th value, network is complete

User Input

- User gives their own data value
- Uses network to predict a value of 1 or 0
- Displays success probability, nodes (input dimension), and execution time



Task: Create a machine learning program that can predict an outcome

Regular Goals

- Read a bunch of sample inputs and results that have known correct values
- Try to create a network that accommodates for most of those data inputs
- Iterates through all sample examples many times and adjusts the network each time using simple backward propagation by changing a bias factor in the calculations
- Take in a user input data value and use a network to predict the value in binary

Stretch Goals

- With the bias factor, add an array of weights to affect final output
- Instead of just editing margin of error, use calculus functions to find the derivative (user gets to select which one)
- Learn and implement threading techniques
- Add another set of weights, maybe in two dimensions
- User can input in any dimension he or she wants
- Implement a real life example of how it would be used the calculate the chance of diseases (such as cholesterol HDL being used for heart disease), or predict motion of a self-driving car (such as angle of steering needed to avoid obstacles)