FINAL PROJECT

DIGIT RECOGNITION NEURAL NETWORK PROJECT

Rishabh Shrivastava, Piyush Prashant

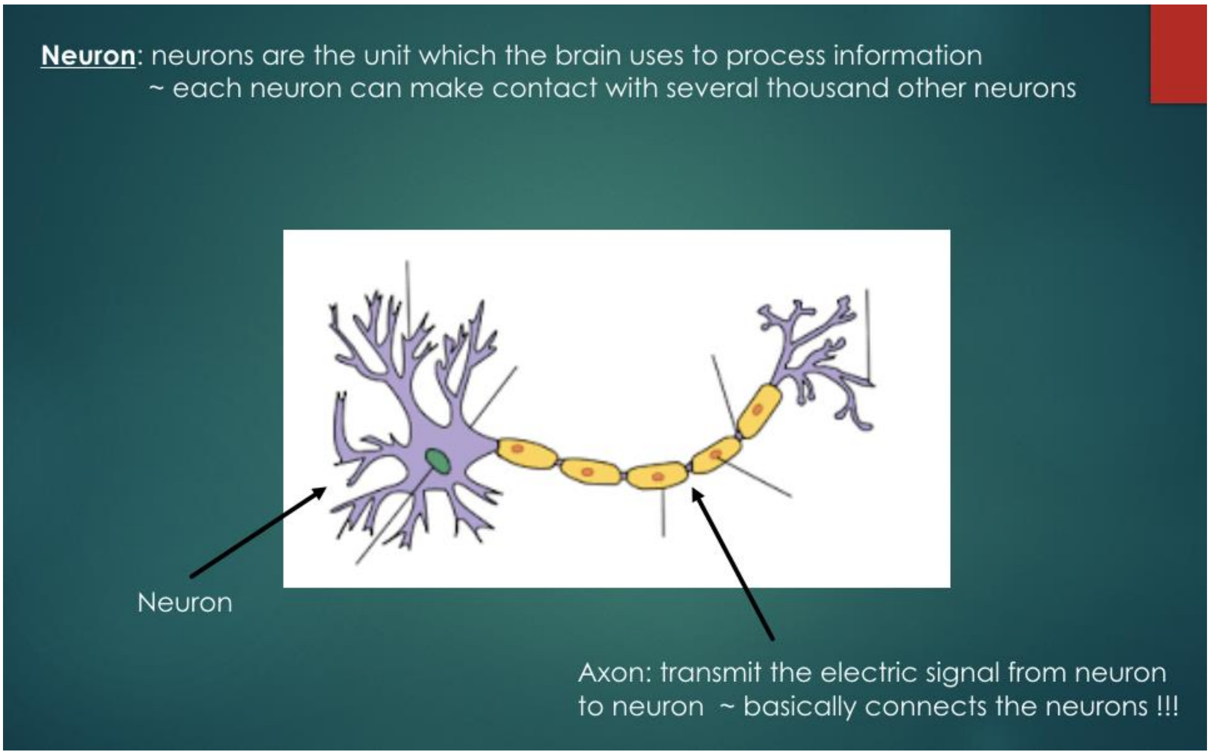
Project Description

This project is a Java-based neural network to analyse the OCR dataset, which is a 8*8 pixel image dataset of handwriting number. Each image only contains one number, between 0 and 9 and the goal of this project is to use the neural network to predict what is the number in the image. By setting up the neural network, use the network to train the dataset and predict the dataset.

Project GitHub links

https://github.com/rishr/NeuralNetworks

Neural Network structure



Neuron: neurons are the unit which the brain uses to process information
~ each neuron can make contact with several thousand other neurons

Neuron

Axon: transmit the electric signal from neuron to neuron ~ basically connects the neurons !!!

Inspired by the biological neural networks.We represent each neuron with a node→ it is basically a directed/undirected graph.Their design enables hem to process information similar way to our biological brains.

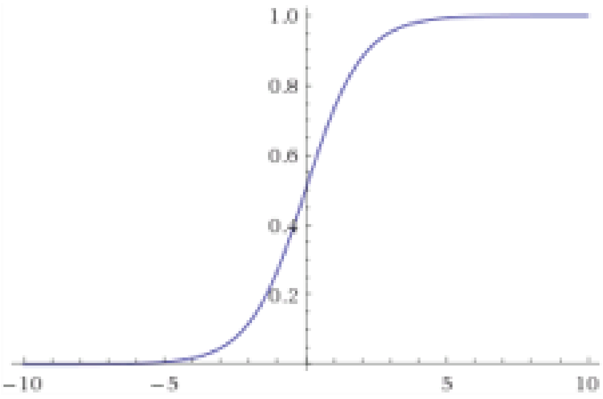
Input layer: We keep feeding our network with data through the input layer.

Hidden layer: It is needed in order to make predictions when we have a non-linear problems.

Output layer: We have the results here.The predicted digit.

Percepton: It is a neuron in the neural network**.** We have to sum up the weighted inputs. For perceptrons we use the step function with a given threshold. It is not working fine for neural network training because the output can be 0 or 1 for perceptrons .A small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip (0 1) That flip may then cause the behaviour of the rest of the network to completely change in some very complicated way

Sigmoid neuron: very similar to perceptrons**.** But small changes in the edge weights causes small change in the output. The inputs / outputs can take any values between 0 and 1 !!!



“Sigmoid function”

Layer

One layer consists of many neurons and can be represented as the following picture:

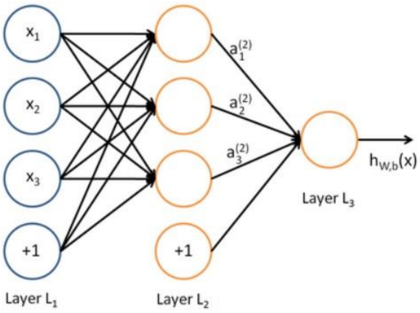
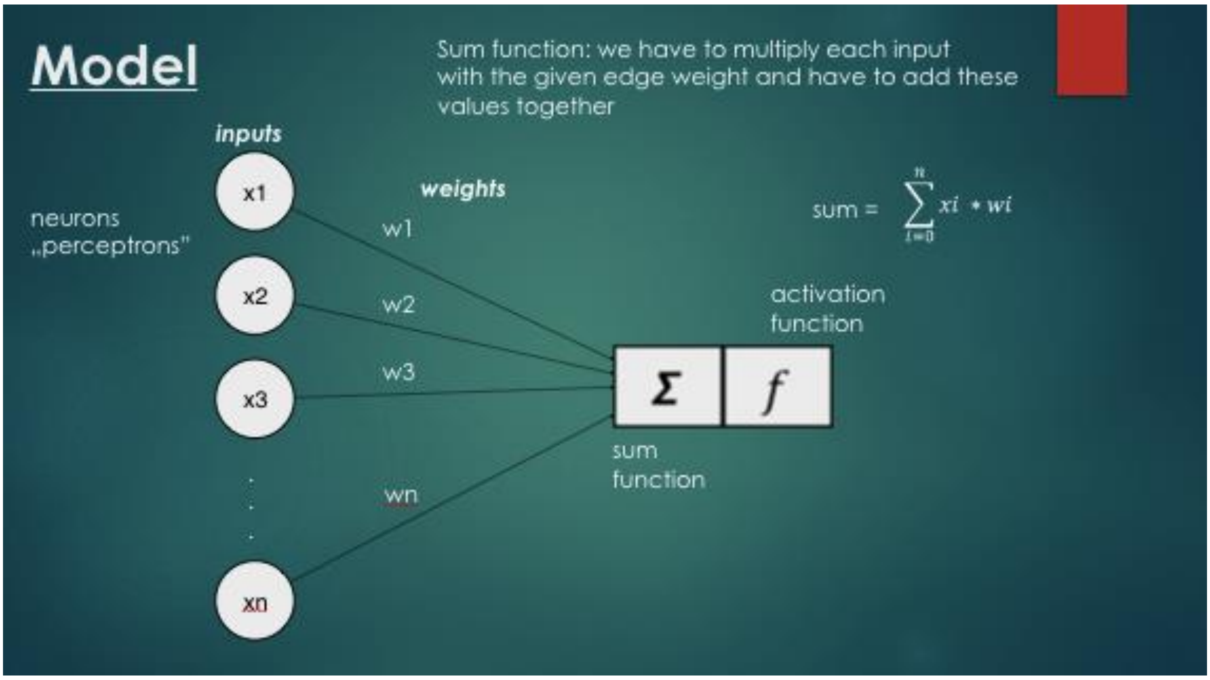


Figure 2. The construction of a neural network layer.



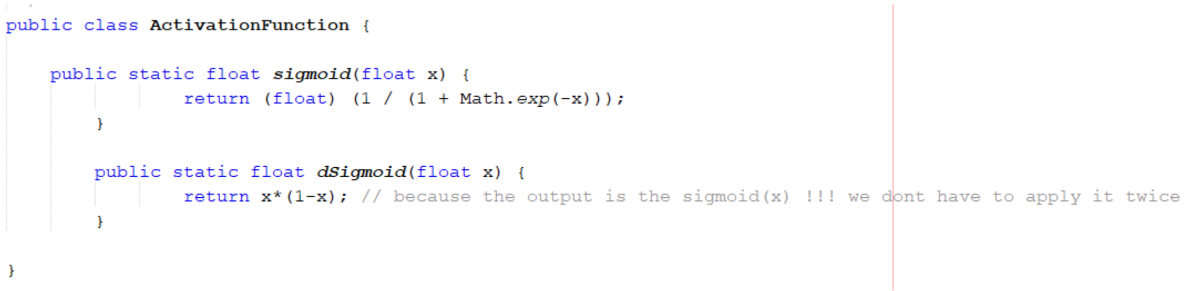
Model

Sum function: we have to multiply each input with the given edge weight and have to add these values together

$$\text{sum} = \sum_{i=0}^{n} xi * wi$$

Neural Network Explanation along with code snippets:

1. Activation Function (Sigmoid Function):



```
public class ActivationFunction {

    public static float sigmoid(float x) {
            return (float) (1 / (1 + Math.exp(-x)));
        }

        public static float dSigmoid(float x) {
            return x*(1-x); // because the output is the sigmoid(x) !!! we dont have to apply it twice
        }

}
```

$$\varepsilon(x) = \frac{1}{1 + e^{-x}}$$

2. Calculating the output:

a. **Calculate the sum**:

$$x_n^l = b_n^l + \sum_k w_{kn}^l \, O_k^{l-1}$$

k = amount of previous neurons

$l = layer$

$p = previous\ neuron$

$n = neuron\ in\ layer\ l$

b. **The desired output is the Sigmoid Value of the sum**:

$$O_n^l = \varepsilon\left(x_n^l\right)$$

Calculating the error to update the weights:

$$E = \tfrac{1}{2}(t - y)^2,$$

where

E is the squared error,

t is the target output for a training sample, and

y is the actual output of the output neuron.

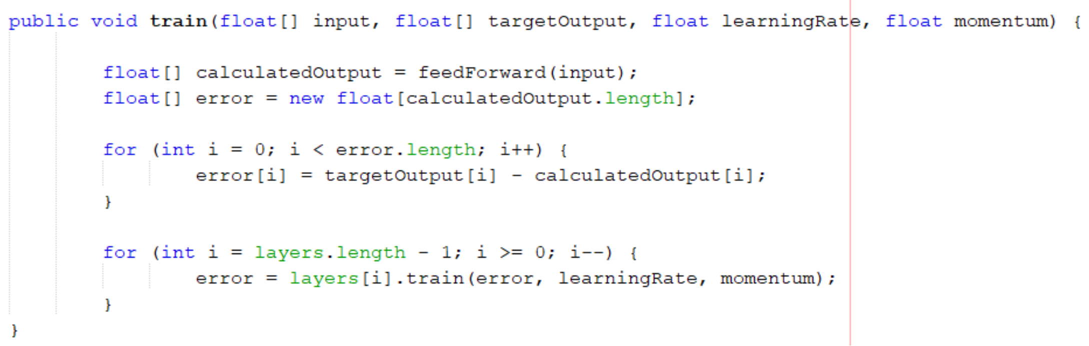
Calculation delta weight (i.e. change in weight) to update the weights to train the network:

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j)o_j(1-o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{\ell\in L}\delta_\ell w_{j\ell})o_j(1-o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

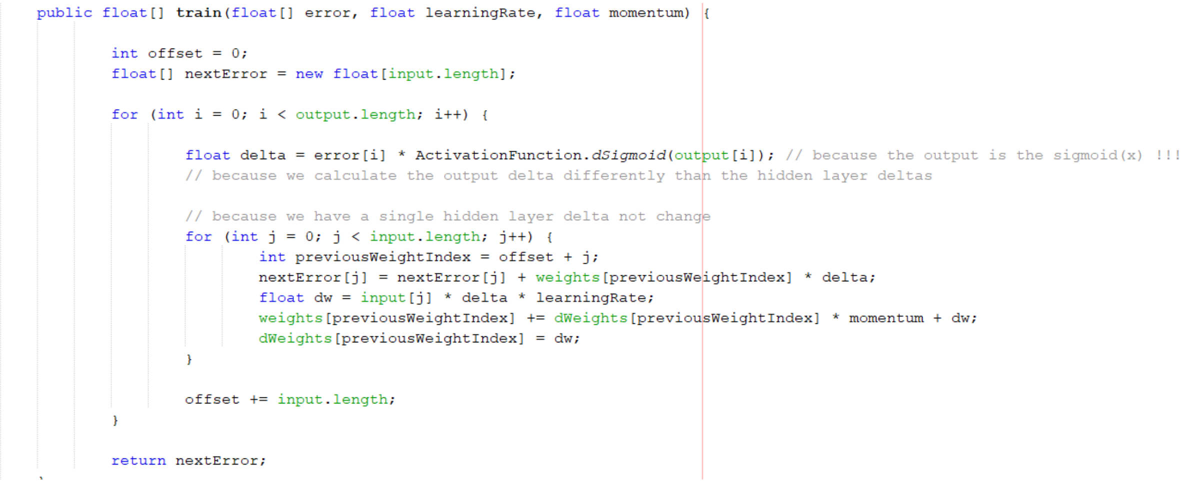
$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = \begin{cases} -\eta o_i(o_j - t_j)o_j(1-o_j) & \text{if } j \text{ is an output neuron,} \\ -\eta o_i\left(\sum_{\ell\in L}\delta_\ell w_{j\ell}\right)o_j(1-o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

Where η = learning rate which should be greater than 0.

Back Propagation Code Snippets:



```
public void train(float[] input, float[] targetOutput, float learningRate, float momentum) {

        float[] calculatedOutput = feedForward(input);
        float[] error = new float[calculatedOutput.length];

        for (int i = 0; i < error.length; i++) {
                error[i] = targetOutput[i] - calculatedOutput[i];
        }

        for (int i = layers.length - 1; i >= 0; i--) {
                error = layers[i].train(error, learningRate, momentum);
        }
}
```



```
public float[] train(float[] error, float learningRate, float momentum) {

        int offset = 0;
        float[] nextError = new float[input.length];

        for (int i = 0; i < output.length; i++) {

                float delta = error[i] * ActivationFunction.dSigmoid(output[i]); // because the output is the sigmoid(x) !!!
                // because we calculate the output delta differently than the hidden layer deltas

                // because we have a single hidden layer delta not change
                for (int j = 0; j < input.length; j++) {
                        int previousWeightIndex = offset + j;
                        nextError[j] = nextError[j] + weights[previousWeightIndex] * delta;
                        float dw = input[j] * delta * learningRate;
                        weights[previousWeightIndex] += dWeights[previousWeightIndex] * momentum + dw;
                        dWeights[previousWeightIndex] = dw;
                }

                offset += input.length;
        }

        return nextError;
}
```

WORKING

Important Classes:

1. Backpropogation.java

This class contains layer initialization and does the feed forward calculation of outputs starting from the input layer until the output layer.

2. Layer.java

This class contains all the input, output, weights , biases, delta weights for all the layers respectively. It is also responsible for the weights initialization. Here the delta weights and activation functions are calculate mainly for each layer.

3. NeuralNetwork.java

This is our main class where we initiate the date loading , backpropagation and its neural network training. Here we calculate the outputs.

4. LoadDataSet.java

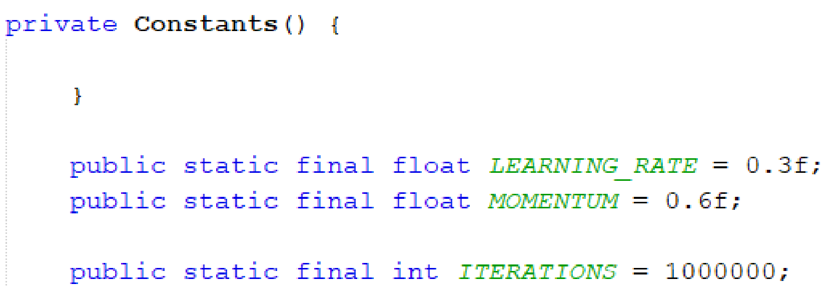
This class is used to loads the pictures and does their pixel calculation.

5. ActivationFunction.java

This is a class that calculates the activation function and the and function which calculates partial derivative of the sigmoid function.

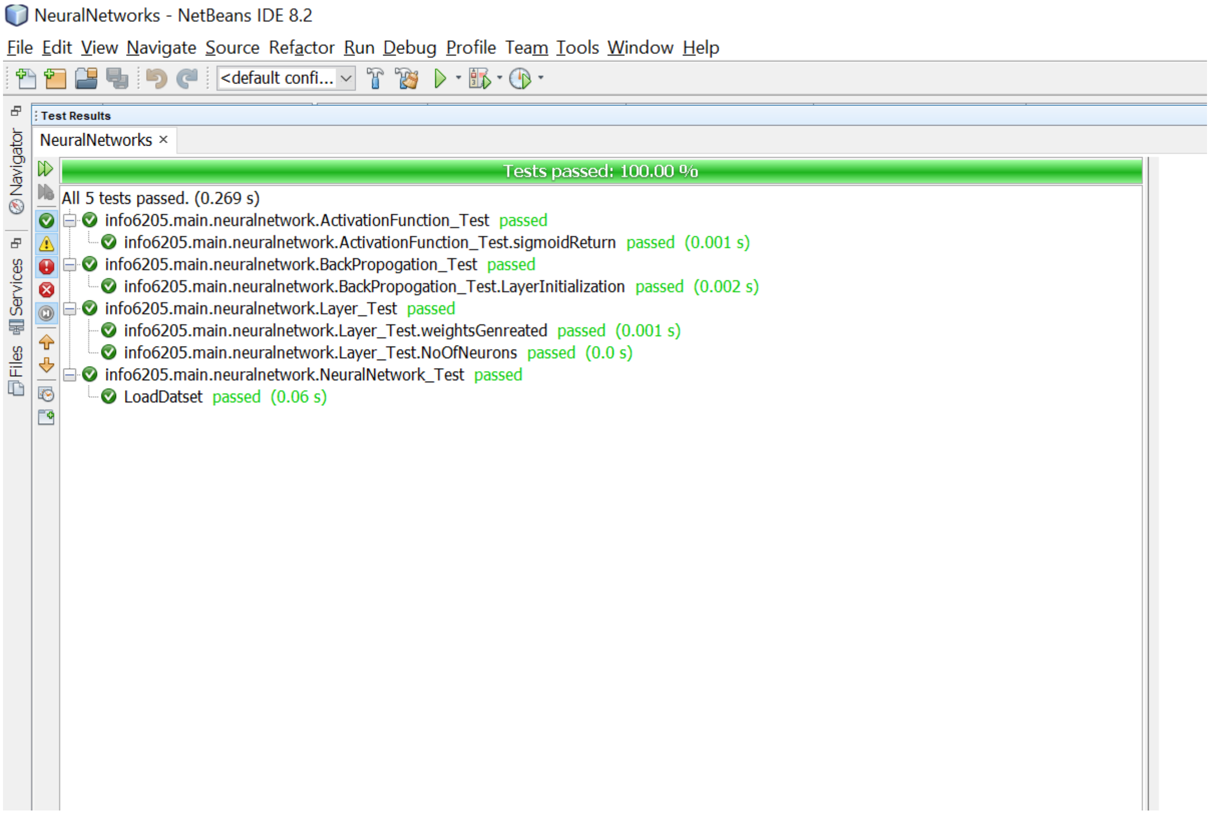
6. Constants.java

This class contains all the constant values for our program such as learning rate, momentum an number of iterations.



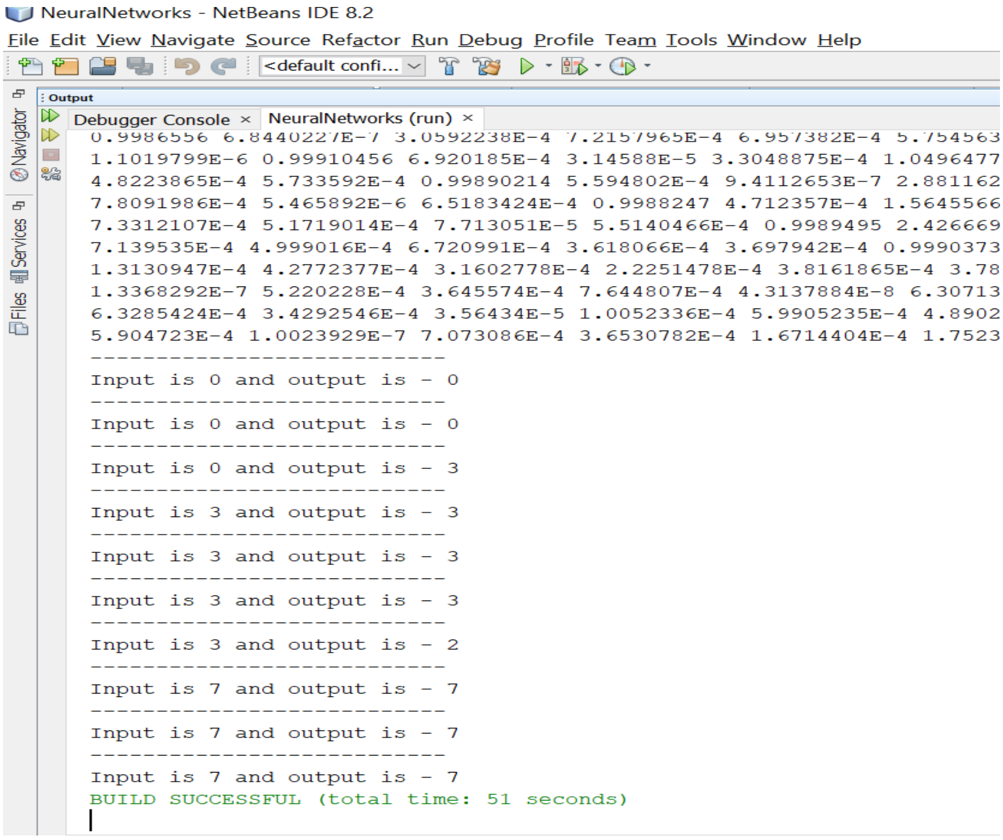
```
private Constants() {

    }

    public static final float LEARNING_RATE = 0.3f;
    public static final float MOMENTUM = 0.6f;

    public static final int ITERATIONS = 1000000;
```

Unit test screenshot:


NeuralNetworks - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default confi...
Test Results
NeuralNetworks ×
Navigator
Services
Files

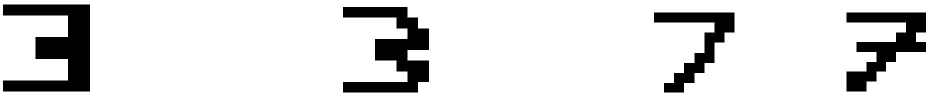
```
Tests passed: 100.00 %
All 5 tests passed. (0.269 s)
info6205.main.neuralnetwork.ActivationFunction_Test  passed
    info6205.main.neuralnetwork.ActivationFunction_Test.sigmoidReturn  passed  (0.001 s)
info6205.main.neuralnetwork.BackPropogation_Test  passed
    info6205.main.neuralnetwork.BackPropogation_Test.LayerInitialization  passed  (0.002 s)
info6205.main.neuralnetwork.Layer_Test  passed
    info6205.main.neuralnetwork.Layer_Test.weightsGenreated  passed  (0.001 s)
    info6205.main.neuralnetwork.Layer_Test.NoOfNeurons  passed  (0.0 s)
info6205.main.neuralnetwork.NeuralNetwork_Test  passed
    LoadDatset  passed  (0.06 s)
```

Output-:


NeuralNetworks - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default confi...
Output
Debugger Console ×
NeuralNetworks (run) ×
Navigator
Services
Files

```
0.9986556 6.8440227E-7 3.0592238E-4 7.2157965E-4 6.957382E-4 5.754563
1.1019799E-6 0.99910456 6.920185E-4 3.14588E-5 3.3048875E-4 1.0496477
4.8223865E-4 5.733592E-4 0.99890214 5.594802E-4 9.4112653E-7 2.881162
7.8091986E-4 5.465892E-6 6.5183424E-4 0.9988247 4.712357E-4 1.5645566
7.3312107E-4 5.1719014E-4 7.713051E-5 5.5140466E-4 0.9989495 2.426669
7.139535E-4 4.999016E-4 6.720991E-4 3.618066E-4 3.697942E-4 0.9990373
1.3130947E-4 4.2772377E-4 3.1602778E-4 2.2251478E-4 3.8161865E-4 3.78
1.3368292E-7 5.220228E-4 3.645574E-4 7.644807E-4 4.3137884E-8 6.30713
6.3285424E-4 3.4292546E-4 3.56434E-5 1.0052336E-4 5.9905235E-4 4.8902
5.904723E-4 1.0023929E-7 7.073086E-4 3.6530782E-4 1.6714404E-4 1.7523
---------------------------
Input is 0 and output is - 0
---------------------------
Input is 0 and output is - 0
---------------------------
Input is 0 and output is - 3
---------------------------
Input is 3 and output is - 3
---------------------------
Input is 3 and output is - 3
---------------------------
Input is 3 and output is - 3
---------------------------
Input is 3 and output is - 2
---------------------------
Input is 7 and output is - 7
---------------------------
Input is 7 and output is - 7
---------------------------
Input is 7 and output is - 7
BUILD SUCCESSFUL (total time: 51 seconds)
```

For testing we input 10 digits to identify with different handwriting styles such as –



The results were -:

S. No	Input	Predicted Output
1	0	0
2	0	0
3	0	3
4	3	3
5	3	3
6	3	3
7	3	2
8	7	7
9	7	7
10	7	7

Correct Predictions = 8
Incorrect Predictions = 2

Prediction Accuracy = 80%

Refernces-:

1. https://www.youtube.com/watch?v=aircAruvnKk
2. https://www.youtube.com/watch?v=QJoa0JYaX1I
3. https://365datascience.com/backpropagation/
4. https://towardsdatascience.com/neural-network-architectures-156e5bad51ba
5. http://mathworld.wolfram.com/SigmoidFunction.html