

DIGIT RECOGNITION

Implemented using Neural Networks

- Astha Sharma(001441488)
- Aishwarya Prabhu(001473651)

TABLE OF CONTENT

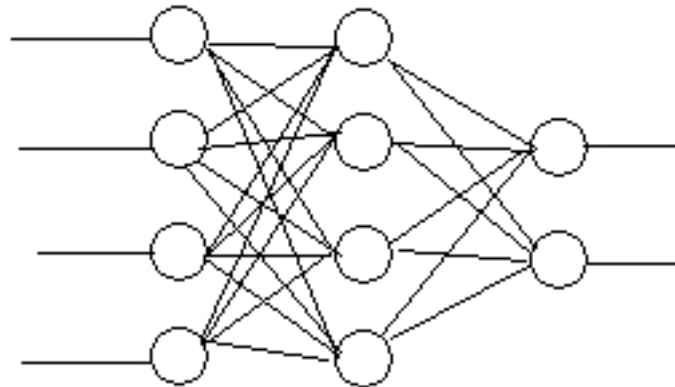
NAME

1. Introduction.....	
2. Problem Statement.....	
3. Implementation Details.....	
4. Program Output.....	
5. Test Cases Passed.....	
6. Conclusion.....	
7. References.....	

INTRODUCTION

One fascinating thing about artificial neural networks is that, they are mainly inspired by the human brain. This doesn't mean that Artificial Neural Networks are *exact* simulations of the biological neural networks inside our brain - because the actual working of human brain is still a mystery. The concept of artificial neural networks emerged in its present form our very limited understanding about our own brain

A neural network consists of several layers, and each layer has a number of neurons in it. Neurons in one layer is connected to multiple or all neurons in the next layer. Input is fed to the neurons in input layer, and output is obtained from the neurons in the last layer.



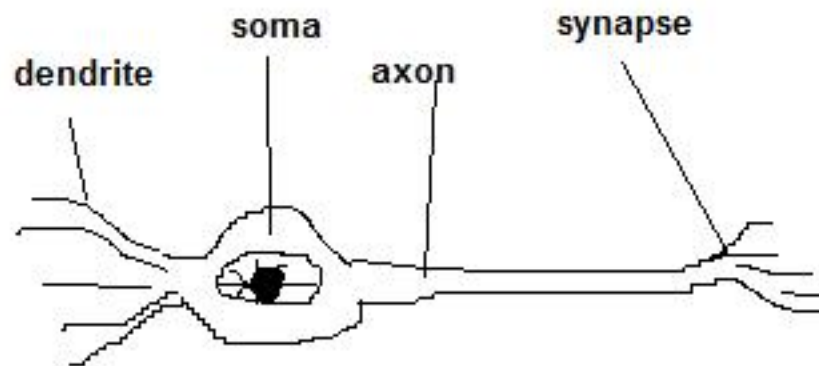
A Fully Connected 4-4-2 neural network with 4 neurons in input layer, 4 neurons in hidden layer and 2 neurons in output layer.

An artificial neural network can *learn* from a set of samples.

For training a neural network, first we provide a set of inputs and outputs. For example, if we need a neural network to detect fractures from an X-Ray of a bone, first we train the network with a number of samples. We provide an X-Ray, along with the information that whether that particular X-Ray has a fracture or not. After training the network a number of times with a number of samples like this (probably thousands of samples), it is assumed that the neural network can 'detect' whether a given X-Ray indicates a fracture in the bone (This is just an example). The basic component in a neural network is a neuron.

Biological Neurons

A biological neuron looks like this:



does this look like a neuron?

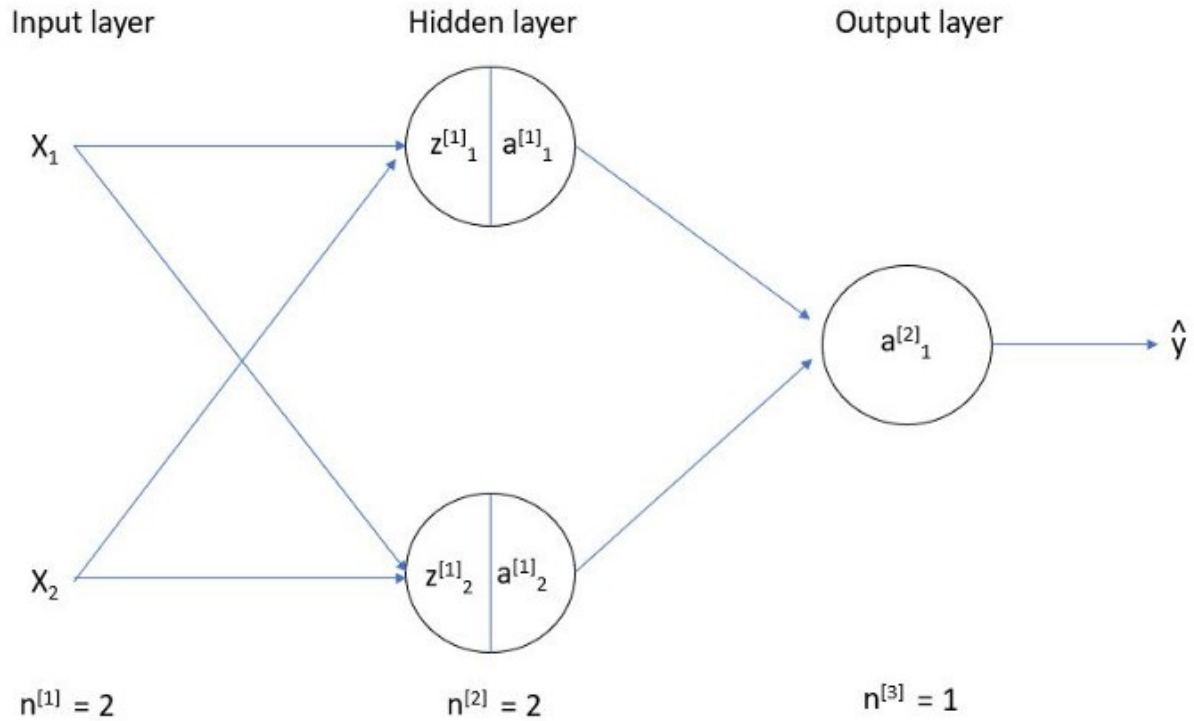
The four basic components of a biological neuron are

- **Dendrites** - Dendrites are hair like extensions of a neuron, and each dendrite can bring some input to the neuron (from neurons in the previous layer). These inputs are given to the soma.
- **Soma** - Soma is responsible for processing these inputs, and the output is provided to other neurons through the axon and synapses.
- **Axon** - The axon is responsible for carrying the output of soma to other neurons, through the synapses
- **Synapses** - Synapses of one neuron is connected to the dendrites of neurons in the next layer. The connections between neurons is possible because of synapses and dendrites.

A single neuron is connected to multiple neurons (mostly, all neurons) in the next layer. Also, a neuron in one layer can accept inputs from more than one neuron (mostly, all neurons) in the previous layer.

Network Architecture

This is a two-layer feed-forward neural network :



Part 1: Forward propagation Equations

Here, in a 2-layer neural network, each input node is connected with each hidden node. So, we calculate the forward propagation equations by using this formula:

$$\mathbf{Z} = \text{Weights} * \text{Input} + \text{bias}$$

$$\mathbf{z}^{[1]}_1 = \mathbf{w}^{[1]}_{11}x_1 + \mathbf{w}^{[1]}_{12}x_2 + \mathbf{b}^{[1]}_1$$

$$\mathbf{a}^{[1]}_1 = \sigma(\mathbf{z}^{[1]}_1)$$

$$\mathbf{z}^{[1]}_2 = \mathbf{w}^{[1]}_{21}x_1 + \mathbf{w}^{[1]}_{22}x_2 + \mathbf{b}^{[1]}_2$$

$$\mathbf{a}^{[1]}_2 = \sigma(\mathbf{z}^{[1]}_2)$$

$$\mathbf{z}^{[2]}_1 = \mathbf{w}^{[2]}_{11}\mathbf{a}^{[1]}_1 + \mathbf{w}^{[2]}_{12}\mathbf{a}^{[1]}_2 + \mathbf{b}^{[2]}_1$$

$$\mathbf{y} = \sigma(\mathbf{z}^{[2]}_1)$$

Now, we vectorize our equations:

$$\begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \end{pmatrix} = \begin{pmatrix} W_{11}^{[1]} & W_{12}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \end{pmatrix}$$

The above matrix can be refactored into matrix multiplication below.

$$\begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \end{pmatrix} = \begin{pmatrix} W_{11}^{[1]} & W_{12}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \end{pmatrix}$$

From this point we have managed to vectorize our NN to accommodate two nodes.

$$\begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \end{pmatrix} \text{ Can be represented as a single vector of shape (nodes, 1) for this case we have 2 nodes.}$$

The above equation can be generalized to:

$$\mathbf{Z} = \mathbf{W} * \text{TRANSPOSE}(\mathbf{X}) + \mathbf{Bias};$$

Activation Function:

We need activation functions to learn non-linear complex functional mappings between the inputs and target outputs of our data. We have used a Sigmoid function in our application. Passing the above output(Z) in the sigmoid function:

$$\text{Sigmoid}(\mathbf{Z}) = 1 / (1 + e^{-\mathbf{Z}})$$

Now, we do the **Backpropagation**:

We try to adjust the weights to get the output close to expected output. The weights of the output layer are adjusted followed by the hidden layers and then the function is trained again in order to get more accuracy.

$$dW = \frac{dJ}{dw} = \frac{dJ}{da} * \frac{da}{dz} * \frac{dz}{dw}$$

$$\frac{dz}{dw} = \frac{d(wx+b)}{dw} = X \quad \dots \dots \dots eq1$$

$$\frac{da}{dz} = \frac{d\left(\frac{1}{1+e^{-z}}\right)}{dz}$$

$$\frac{da}{dz} = \frac{d\left(\frac{1}{1+e^{-z}}\right)}{dz} = \frac{d(1+e^{-z})^{-1}}{dz} = -1(1+e^{-z})^{-2} * (-e^{-z}) = (1+e^{-z})^{-2} * (e^{-z})$$

$$\frac{da}{dz} = \frac{(e^{-z})}{(1+e^{-z})^2} = \frac{(1+e^{-z})^{-1}}{(1+e^{-z})^2} = \left(1 - \frac{1}{1+e^{-z}}\right) \frac{1}{1+e^{-z}} = (1-a)a \quad \dots \dots \dots eqn2$$

$$\frac{dJ}{da} = \frac{d(-y \log(a) - (1-y) \log(1-a))}{da} = -\frac{y}{a} + \frac{1-y}{1-a} \quad \dots \dots \dots eqn3$$

Combining equations 1, 2 and 3

$$dW = \frac{dJ}{dw} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) * (1-a)a * x = x(a-y)$$

$$dW = \frac{dJ}{dw} = (A - Y)x$$

So, we get :

$$W = W - \alpha dW$$

$$b = b - \alpha db$$

2. PROBLEM STATEMENT

The XOR problem is solved by the 2 layer perceptron. Given, all 4 boolean inputs and outputs, it trains and memorizes the weights needed to reproduce the I/O.

The XOR table is as follows:

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

So, we feed the X and Y inputs, train the network to give the correct output according to the XOR operations performed on the input set.

We have taken 2,2,1 as our topology for the neural network.

After training the network for 4000 times, the network gives up to 89% accuracy in outputs. The cost of the network is greatly reduced after training it for 4000 times.

3. IMPLEMENTATION DETAILS

Feed forward algorithm **pseudocode:**

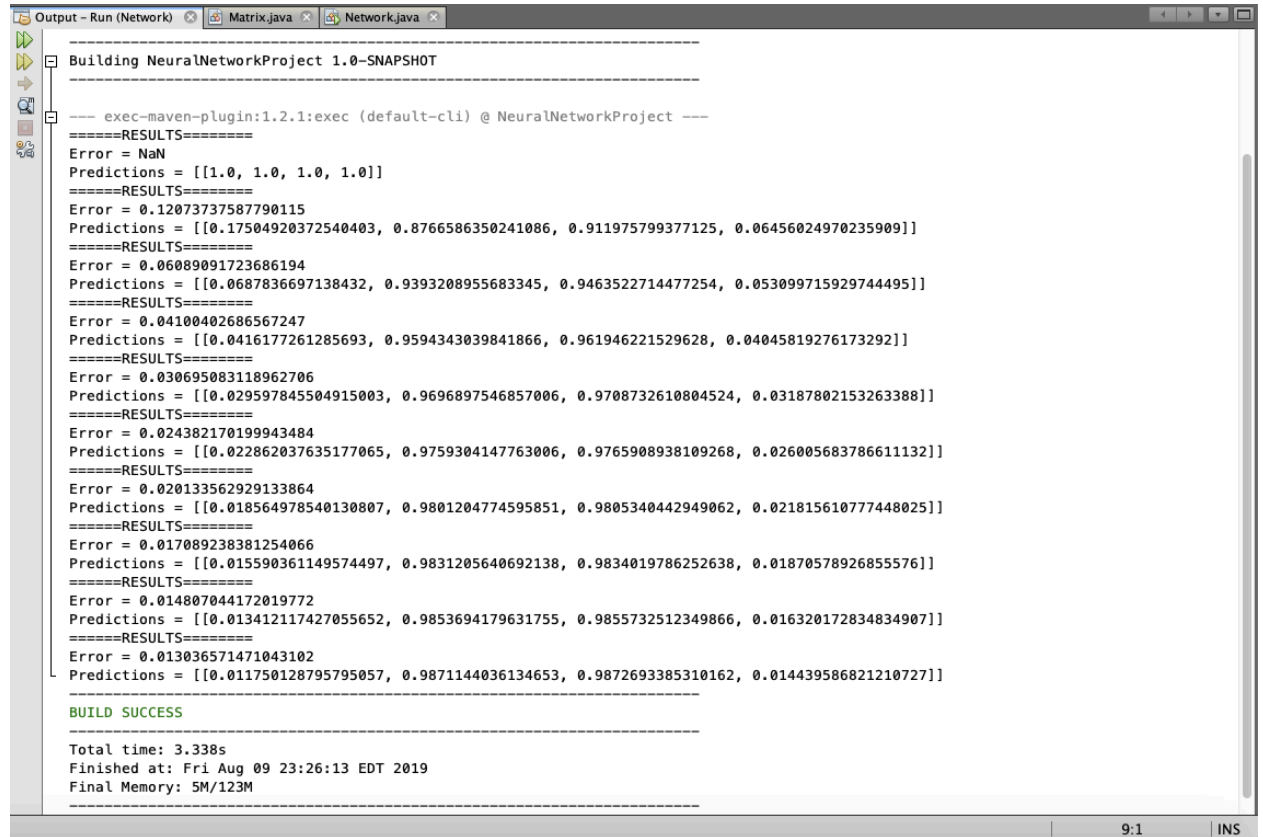
```
function feedForward() {  
    Step 1. Do this while total layers>0  
    Step 2. Calculate random weights  
    Step 3. Calculate random biases  
    Step 4. Calculate sum of (Weights*input)+bias  
    Step 5. Pass the above sum in the sigmoid function  
    Step 6. Take the output from the previous layer as input for next layer  
    Step 7. Goto Step 1  
}
```

Back propagation algorithm **pseudocode:**

```
function backpropagation() {  
    Step 1. Calculate Error: Target-Actual Output  
    Step 2. Calculate gradient by taking differential of sigmoid  
    Step 3. Apply the formula: learning rate*(o/p+(1-o/p)) *Transpose(H)  
    Step 4. Adjust weights and biases accordingly  
}
```


4. PROGRAM OUTPUT

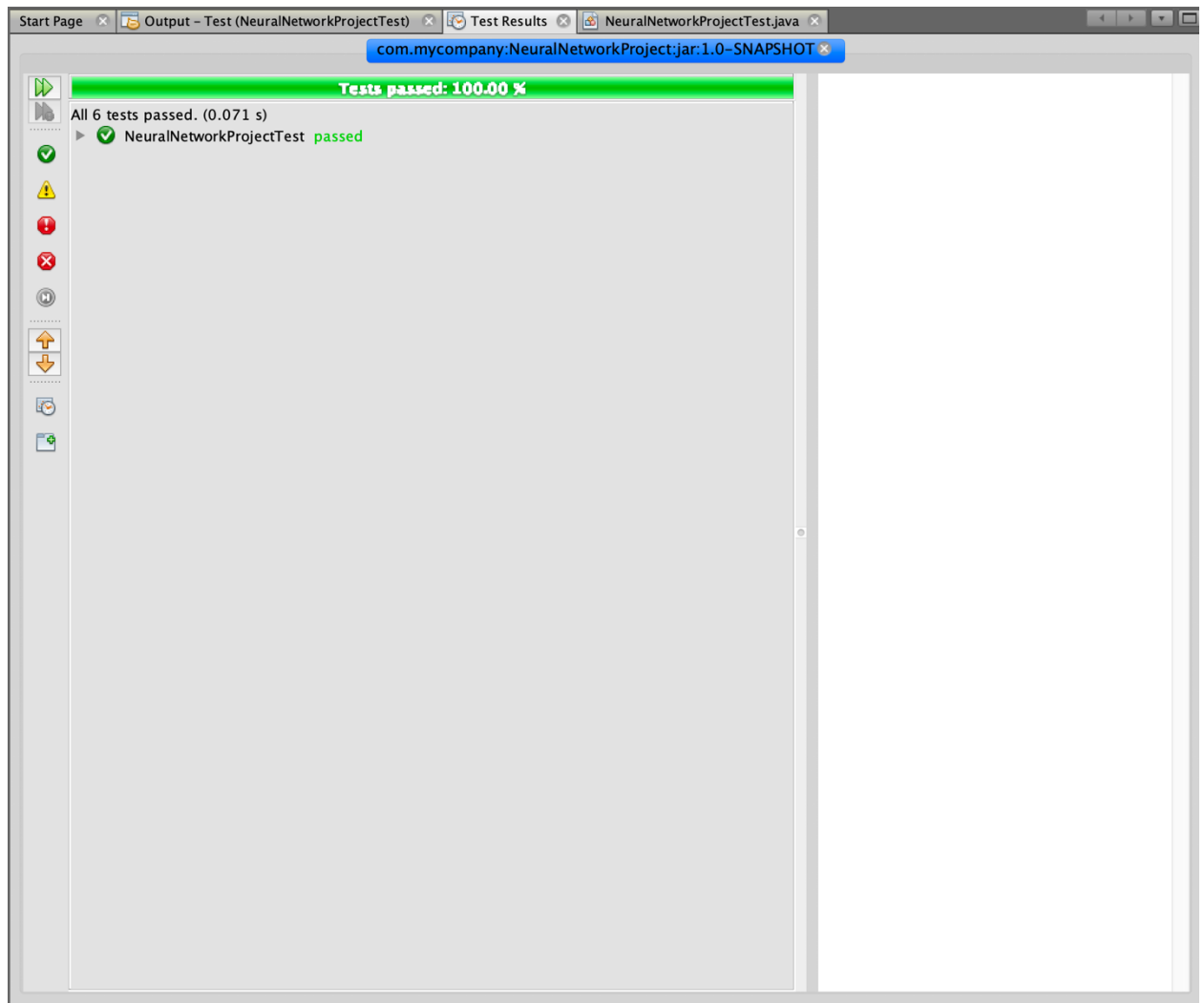
The below program outputs display the reduction in cost with increased iterations. We have printed the cost which is basically the calculated cross entropy. Also, we have printed the outputs that our network prints according to the input that is fed.



```
Output - Run (Network) Matrix.java Network.java
-----
Building NeuralNetworkProject 1.0-SNAPSHOT
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ NeuralNetworkProject ---
=====RESULTS=====
Error = NaN
Predictions = [[1.0, 1.0, 1.0, 1.0]]
=====RESULTS=====
Error = 0.12073737587790115
Predictions = [[0.17504920372540403, 0.8766586350241086, 0.911975799377125, 0.06456024970235909]]
=====RESULTS=====
Error = 0.06089091723686194
Predictions = [[0.0687836697138432, 0.9393208955683345, 0.9463522714477254, 0.053099715929744495]]
=====RESULTS=====
Error = 0.04100402686567247
Predictions = [[0.0416177261285693, 0.9594343039841866, 0.961946221529628, 0.04045819276173292]]
=====RESULTS=====
Error = 0.030695083118962706
Predictions = [[0.029597845504915003, 0.9696897546857006, 0.9708732610804524, 0.03187802153263388]]
=====RESULTS=====
Error = 0.024382170199943484
Predictions = [[0.022862037635177065, 0.9759304147763006, 0.9765908938109268, 0.026005683786611132]]
=====RESULTS=====
Error = 0.020133562929133864
Predictions = [[0.018564978540130807, 0.9801204774595851, 0.9805340442949062, 0.021815610777448025]]
=====RESULTS=====
Error = 0.017089238381254066
Predictions = [[0.015590361149574497, 0.9831205640692138, 0.9834019786252638, 0.01870578926855576]]
=====RESULTS=====
Error = 0.014807044172019772
Predictions = [[0.013412117427055652, 0.9853694179631755, 0.9855732512349866, 0.016320172834834907]]
=====RESULTS=====
Error = 0.013036571471043102
Predictions = [[0.011750128795795057, 0.9871144036134653, 0.9872693385310162, 0.014439586821210727]]
=====
BUILD SUCCESS
-----
Total time: 3.338s
Finished at: Fri Aug 09 23:26:13 EDT 2019
Final Memory: 5M/123M
-----
```

5. TEST CASES PASSED

We wrote 6 test cases that test the random function, sigmoid function and derivative of sigmoid function.



6. CONCLUSION

The error is () reduced with increase in iterations.

The neural network application is trained to predict the XOR output of the 2 inputs that are fed.

After training the network for 4000 times, the network gives up to 89% accuracy in outputs. The cost of the network is greatly reduced after training it for 4000 times.

The network predicts the output according to the following XOR table:

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

7. REFERENCES

<https://medium.com/coinmonks/implementing-an-artificial-neural-network-in-pure-java-no-external-dependencies-975749a38114>

<https://www.codeproject.com/articles/14342/designing-and-implementing-a-neural-network-librar>

<https://www.youtube.com/watch?v=XJ7HLz9VYz0&list=PLRqWX-V7Uu6aCibgK1PTWWu9by6XFdCfh>

<http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>

<https://dzone.com/articles/java-image-cat-and-dog-recognition-with-deep-neura>