OPEN ACCESS
Saudi Journal of Engineering and Technology
Abbreviated Key Title: Saudi J Eng Technol
ISSN 2415-6272 (Print) |ISSN 2415-6264 (Online)
Scholars Middle East Publishers, Dubai, United Arab Emirates
Journal homepage: https://saudijournals.com

Original Research Article

Solution for XOR Problem with Neural Networks Using Google Colab and MATLAB / Simulink

Viswanatha V[1*], Ramachandra A C[2], Berwyn Suhas[3], Adithya T[3]

[1]Assistant Professor, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, India
[2]Professor, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, India
[3]UG Student, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, India

DOI: 10.36348/sjet.2023.v08i01.003 | **Received:** 08.12.2022 | **Accepted:** 22.01.2023 | **Published:** 26.01.2023

***Corresponding author:** Viswanatha V
Assistant Professor, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, India

Abstract

To find solution for XOR problem we are considering two widely used software that is being relied on by many software developers for their work. The first software is Google Colab tool which can be used to implement ANN programs by coding the neural network using python. This tool is reliable since it supports python language for its implementation. The other major reason is that we can use GPU and TPU processors for the computation process of the neural network. The major advantage of this is that for complex Neural Networks instead of using CPU present in the user's system we can use those two processors through online mode without purchasing such processors for our computation. The next software that can be used for implementing ANN is Matlab Simulink. This software is used for highly calculative and computational tasks such as Control System, Deep Learning, Machine Learning, Digital Signal Processing and many more. Matlab is highly efficient and easy to code and use when compared to any other software. This is because Matlab stores data in the form of matrices and computes them in this fashion. Matlab in collaboration with Simulink can be used to manually model the Neural Network without the need of any code or knowledge of any coding language. Since Simulink is integrated with Matlab we can also code the Neural Network in Matlab and obtain its mathematically equivalent model in Simulink. Also, Matlab has a dedicated tool in its library to implement neural network called NN tool. Using this tool, we can directly add the data for input, desired output, or target. After inserting the required data, we can train the network with the given dataset and receive appropriate graph's which will be helpful in analyzing the data. Once the Neural Network is trained, we can test the network through this tool and verify the obtained results.

Keywords: XOR problem, Back propagation, Feature extraction, Deep learning.

Copyright © 2023 The Author(s): This is an open-access article distributed under the terms of the Creative Commons Attribution **4.0 International License (CC BY-NC 4.0)** which permits unrestricted use, distribution, and reproduction in any medium for non-commercial use provided the original author and source are credited.

1. INTRODUCTION

Artificial Neural Network is a part of Deep Learning which is widely used in this generation as it is the next arising technology with immense scope in the present IT sector. The main reason for developing ANN is to try to simulate the working of a human brain [1]. The Human brain is a complex organ consisting of various networks interconnected to each other through neurons. A neuron is the basic building block of a network on which the neural network works upon. If this vast network of complexity can be utilized by a machine it can perform wonders beyond human imagination [2]. Hence research regarding Neural Network has been initiated. From the diagram shown in fig.1, we can compare a neuron with the block diagram representation of the ANN. The Dendrites acts as the input for the neuron, And the main nucleus is represented as a summer present in the ANN network. The Axons are represented as nodes connecting the next neuron with weights associated with it [3-5].

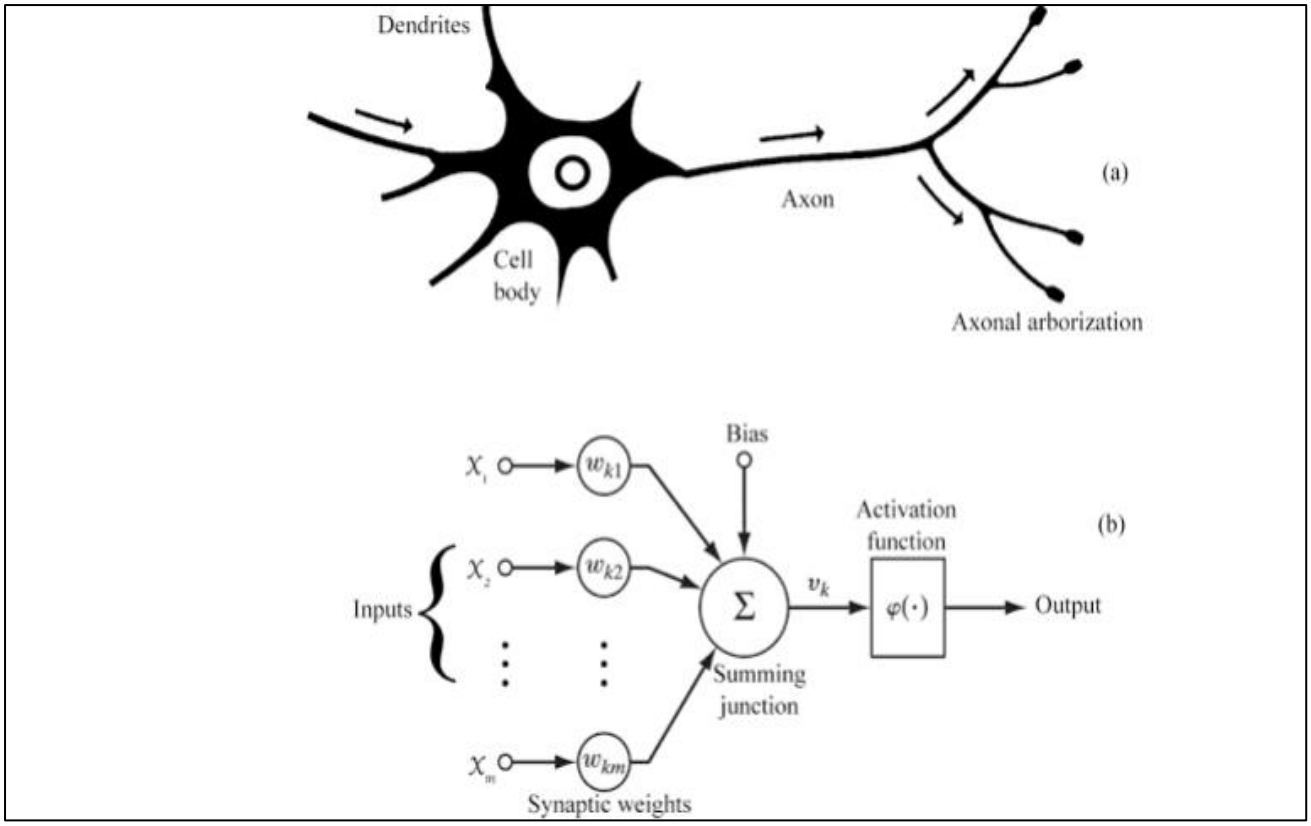

Fig. 1: Neuron with the block diagram representation

In this sector we are going to discuss about implementing XOR gate shown in fig. 2 in the form of a Neural Network. Unlike the other basic gates such as AND, OR and NOT Gates, XOR gate cannot be realised as a neural network easily. This is because of non-linearity condition which is observed in XOR gate when compared to other basic gates [6-8]. All the outputs such as logic 0's and logic 1's of other gates can be separated with a straight line plotted on the graph, but that is not the case for XOR Gate. Hence this gate cannot be realized with a basic neural network containing an input and an output layer. Hence to solve this problem we are using two hidden layer which lies between the input and output neuron. And also, we have to use back propagation algorithm to update the weights and produce the desired output [9-11].

Symbol	Truth Table		
	A	B	Q
A, B, =1, Q 2-input Ex-OR Gate	0	0	0
	0	1	1
	1	0	1
	1	1	0

Fig. 2: XOR gate with functional logic

When we consider basic gates such as AND, OR and NOT gates we can realize these gates and implement them into neural network easily using single layer perception consisting of only input layers and output layers as shown in fig.3 of OR gate. But the same methodology cannot be used to implement an XOR gate as the output of the XOR gate is not linear. Hence the outputs cannot be separated using a straight line as shown in the fig.3 of XOR gate.. To solve this issue we use multilayer perception which consists of input layers, hidden layers and output layers. The hidden layers are used to provide a non-linear property to the neural network. Hence by using two hidden layers we can implement XOR gate in a neural network [12-15].

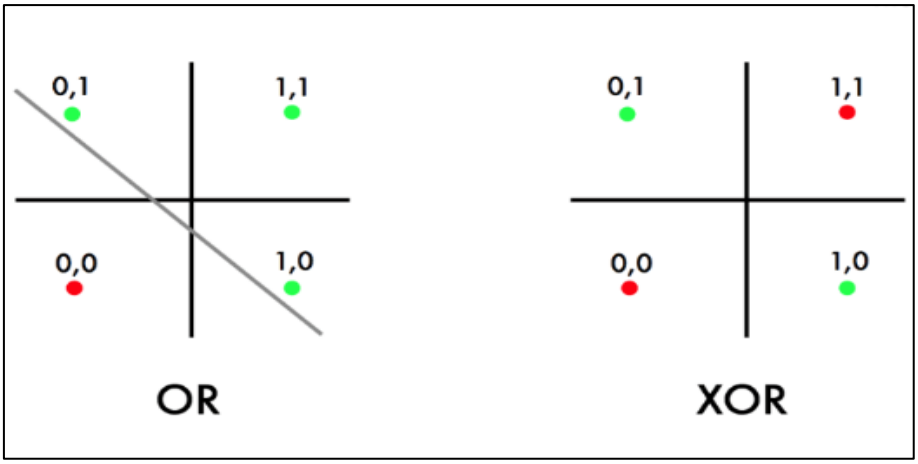

Fig. 3: Data space in plane for OR and XOR gates

2. METHODOLOGY

The neural network architecture of a XOR gate is as shown in fig.4. This network is called forward propagation network as the signal flow traverse from input to output in a linear path. In this network we have two input layers which give out the two inputs of a XOR gate. There are two hidden layers present and they are being provided with a biasing input [B1 & B2] at their respective nodes. These two hidden layers are connected to the output layer in which the output layer is also being provided with a biasing input [B3]. Between the input and hidden layers there are weights attached to each node [W1,W2,W3 &W4] and also between hidden layer and output weights[W5 & W6] are present. As the signal passes through these nodes, they are multiplied by their respective nodes before entering the neuron for further computation. This is a Feed Forward network as there are no feedback loops present to pass the output value back to the input [16, 17].

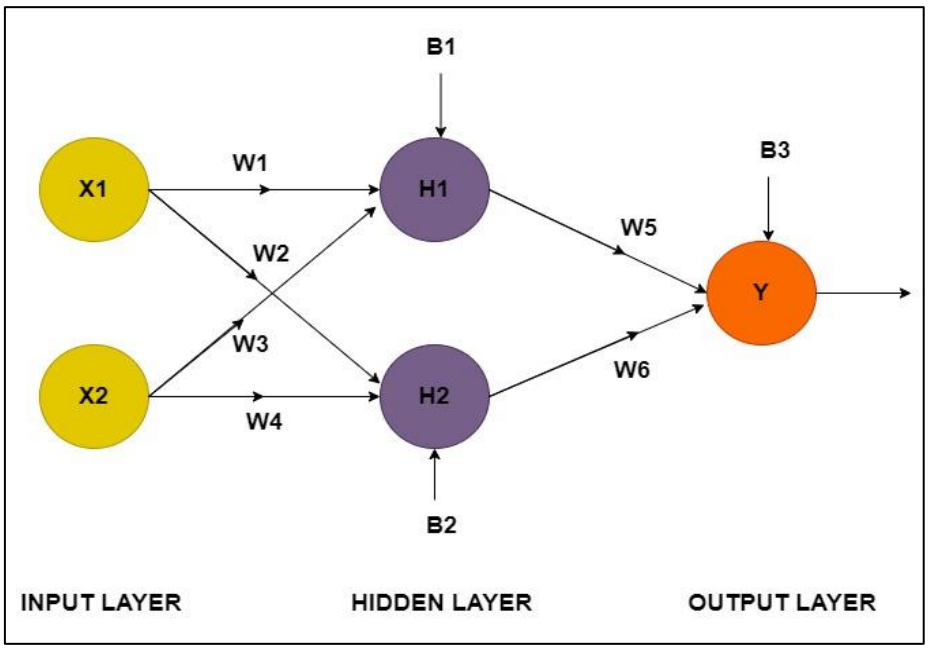

Fig. 4: Neural network architecture of non-linear system as XOR gate

The architecture shown in fig. 5 displays us the application of the back propagation algorithm to Feed Forward XOR Neural Network. Here if the Neural network fails to give out the desired output. Hence, we implement back propagation algorithm to tune the weights. The error signal generated is taken as input and it is used for computation of change in weight. The change in weight must be calculated for each node present in the neural network. Once this change of weight is calculated they must be added to their respective weight at their respective nodes. Once the weights are being updated, we are again testing the network by forward propagation. If the output isn't matching with the desired output still, then we again find error signal and implement back propagation again. This process is continued until the desired output is being achieved from the neural network [18, 19].

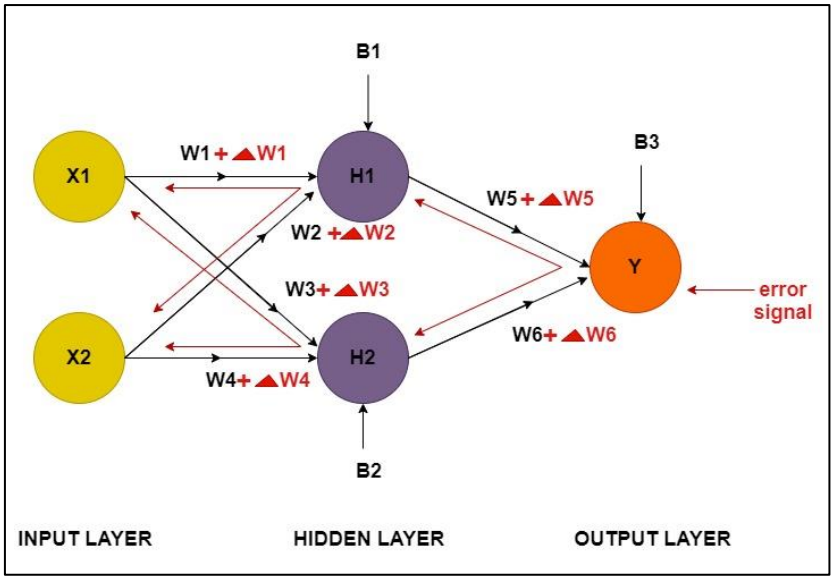

Fig. 5: Feed forward neural network architecture of non-linear system as XOR gate

3. IMPLEMENTATION AND RESULTS

This section shows two different methods of implementing ANN for XOR gate:

3.1). Implementing ANN using Google Colab Using Python

3.2) Implementing ANN using ANN using Matlab Simulink

3.1). ***Implementing ANN using Google Colab Using Python*****:** First open Google Colab from Chrome search or any of its alternatives and create a new file of file type. ipynb. Now before coding we must import some necessary libraries such as NumPy and matplotlib. NumPy is mainly used in python code for working with arrays. Matplotlib is used for plotting necessary graph for better analysis. After importing libraries, we must define the activation function. The activation function that we are using is sigmoid function. After this we start initializing all the parameters such as input and output neurons, hidden layers, weights, bias, epochs, and learning rate. We use rand function which assigns a random value to the weights and assign zero value to the bias input using function zeros. Next step is to use all the parameters for defining forward propagation. Here we multiply the input with weight using dot function and add biasing input. Then we provide the output of this computation to the activation function. In this way the forward propagation from input to the hidden layer has been designed and coded. Now we do the same computation for traversing from the hidden layer to output layer. Now after defining Forward propagation, we will define backward propagation algorithm to update the weights so that the neural network can learn and give the proper output. Now to train the network we provide the inputs of XOR gate and also the desired outputs using NumPy library. After this we set the learning rate of the network and epochs. Epochs is used to provide number of iterations the training should undergo to update the weights and increase the efficiency of the network. Once the training is completed, we shall calculate the efficiency of the network using losses. By plotting losses v/s epochs we understand efficiency of the network increasing as the number of epochs is increased. Epochs are usually kept at higher value to give more iteration to improve neural network. Final step is to print the actual output.

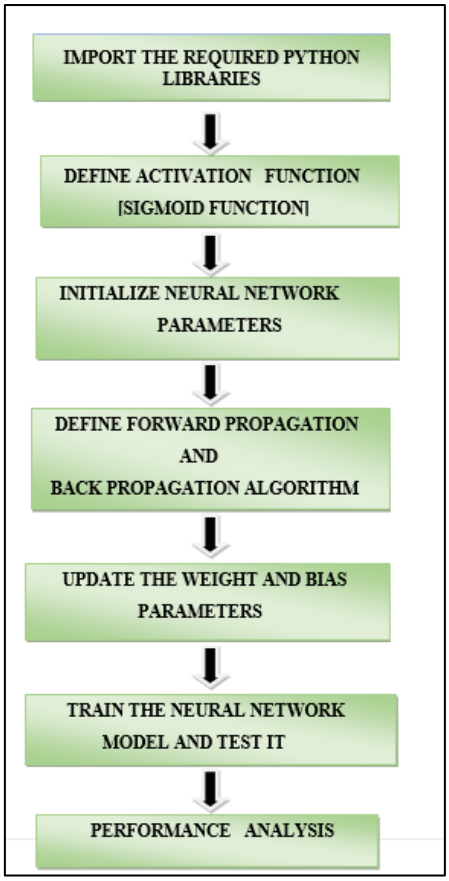

Fig. 6: Flowchart of Methodology used in Google Colab for XOR gate

© 2023 | Published by Scholars Middle East Publishers, Dubai, United Arab Emirates

STEP 1: Import the Required Python Libraries

To create neural network import Numpy and name it as np. Also import matplotlib for plotting the graph for the given data.

STEP 2: Define Activation Function [Sigmoid]

First, we are defining the activation function. We are using sigmoid function which is the most used activation function in neural networks.
FORMULA = 1/(1+exp(u))
Where, u = input data given to the sigmoid function.

STEP 3: Initialize Neural Network Parameters

In this step we are initializing the parameters such as weights and bias to the neural network. W1 is the weight present between input and hidden layer node. W2 is the weight present between hidden layer and output node. b1 and b2 are the biasing inputs given at hidden layer and output layer respectively.

STEP 4: Define Forward Propagation and Back Propagation Algorithm

After defining the parameters, we initialize them while defining forward propagation algorithm. Z1 and Z2 holds the input to hidden layer node multiplied with their associated weights.
FORMULA:
$Zi = \sum Wij*Xi + bi$
Where, Z = summation of input multiplied with their respective weights added with a bias.

After calculating Z1 and Z2 they are given as input to sigmoid function and the final output is stored A1 and A2 respectively. A1 holds the final output value at hidden layer and A2 holds the final output at output layer. Cache is used to store the parameter values and cost function is calculated to find error in actual output obtained.

In back propagation definition, first we have to calculate error signal value which is obtained by subtracting desired output from the actual output. Using this error signal, we are updating the weights and bias inputs using relevant formulas.

STEP 5: Update the Weight And Bias Parameters

After back propagation has been implemented, the gradients obtained from it have to be used to update the parameters.
$W_i = W_i - \eta * \Delta W_i$
Where,
W_i = weight to be updated
ΔW_i = gradient value to be used to update weights
η = learning rate

Once the weights have been updated, we must return them to the variable used to store which is named as Parameter and implement the same in forward propagation algorithm.

STEP 6: Train the Neural Network Model and Test It

Once all parameters and algorithms such as forward and backward propagation algorithm, it is time to call all these functions under training function to train the neural network. Here we are initializing the inputs, desired output, epochs and learning rate to train the model. Once the model is trained, we must test it by providing input via the log tab and verify the output. If the output isnot matching with the desired output then increases number of iterations or epochs to further train the network.

STEP 7: Performance Analysis

To accurately measure the performance of the neural network we are calculating losses. With respect to epochs, we are plotting the graph of epochs V/s losses where losses is set to high at the initial stage before training the model.

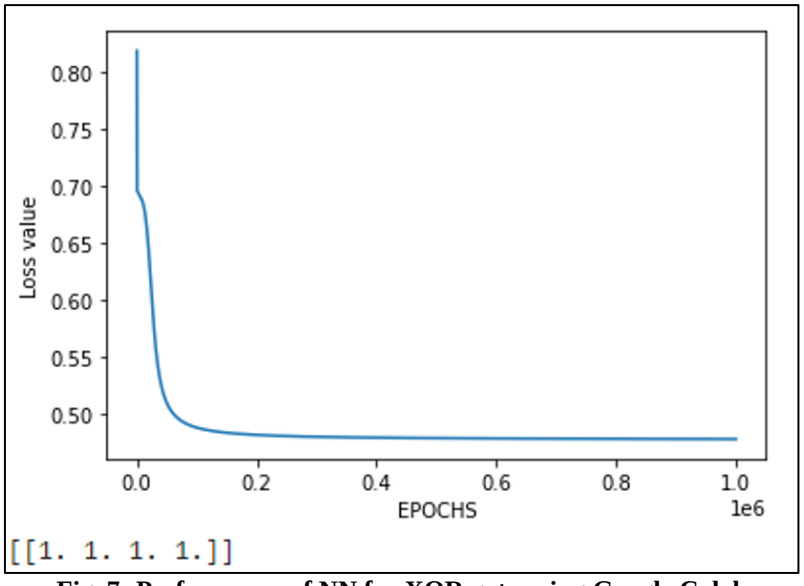

Fig. 7: Performance of NN for XOR gate using Google Colab

From fig.7, we can observe that initially the loss is high, but as the neural network is trained its loss value decreases with increase in epochs.

***3.2). Implementing ANN Using Matlab Simulink*:** There are various ways to implement ANN in Matlab. One among them is Simulink where a neural network can be built hands on using inbuilt library functional blocks which are already predefined in the library. Also, one can code the neural network in Matlab and the same can be converted into Simulink model without creating the model manually. Hence this tool is very versatile when compared to other tools regarding neural network. For our project implementing XOR gate using Simulink has been done manually where we have built an already trained neural network and simulating its functionality in Simulink software. In this model we have two input layers acting as input terminals of an XOR gate. Since XOR gate gives a non-linear output, we are using two hidden layers for compensating this non-linear property. At input neuron, we are connecting clock signal to a switch terminal, when the clock pulse is low output of the switch gives an output of logic '0' and when the clock signal is high, the switch gives an output of logic '1'. These output signals coming from input layer are multiplied by weights which are already tuned to provide the right functionality for the neural network. At the hidden layer these input signals are added by a summer along with a bias input and the output of the summer is given as input to an activation function. The activation function used here is called ReLu function. This function gives a output logic '0' if the input is less than the value '0', and if the input is more than the value '0' then the output is logic '1'. After this process, the hidden layer output is connected to the output layer via a node which is also multiplied by its associated weights and then allowed to enter the output neuron. The same procedure that of the hidden layer is followed in the output layer and the final output will be in terms of either logic'0' or logic '1'. Consider fig. 8, which is representing XOR gate implemented in a neural network using Matlab Simulink software. In this block diagram there are three layers, they are Input Layers, Hidden Layers and one Output Layer. Scope block is predefined in Matlab Simulink which is used for displaying the final output waveform of the XOR gate. Since the value of weights between input layer and hidden layer is 1, we are ignoring them. The weight (w3) between hidden layer 1 and output layer is -2, we are using a multiplier block which is predefined in Matlab Simulink to represent the weight (w3). In input layer there are 2 input neurons, hidden layer there is 2 neuron namely hidden layer 1 and hidden layer 2 and in output layer there is 1 output neuron. These neurons are a subsystem defined in the Simulink library that has their own individual block diagram and its associated waveforms.

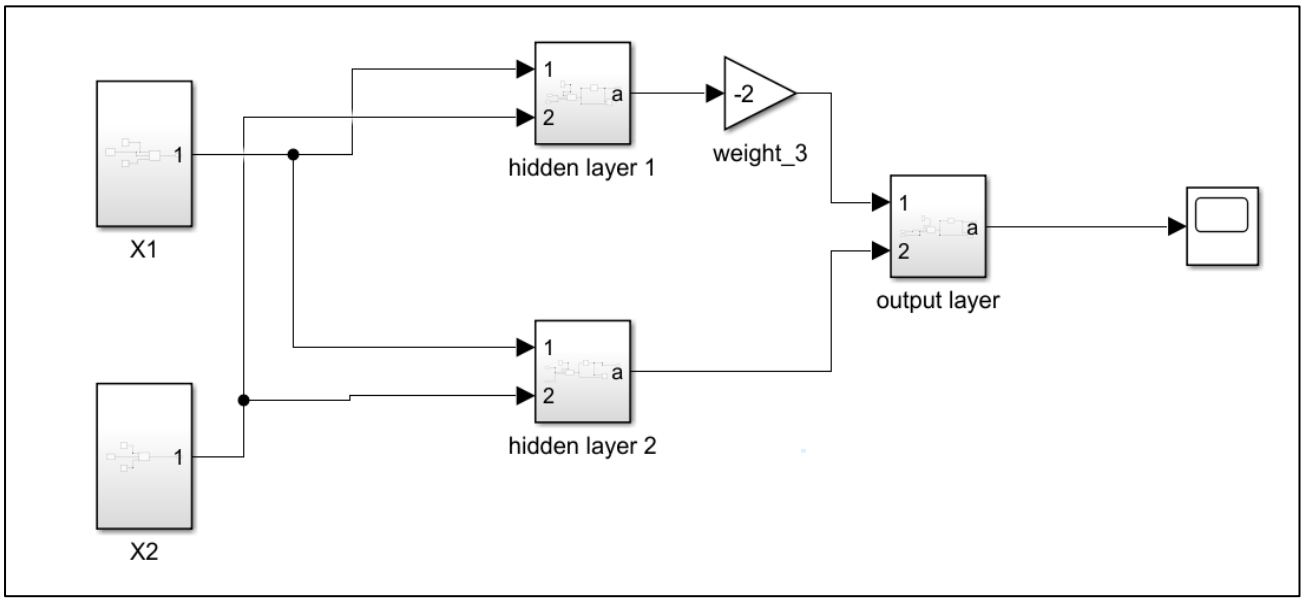

Fig. 8: Neural network of XOR in Simulink

3.2.1). INPUT LAYER NEURON 1

One of the distinct characteristics of the input layer is that artificial neurons in the input layer have a different role. Passive neurons do not take the information from the previous layers because they are the first layer of the neural network. Based upon the design shown in fig.9 for input layer neuron 1, the corresponding output waveform of the input layer is obtained and as shown in fig.10. Here total time duration of the waveform is 10 units and for every 5 units of clock pulses applied waveform of input layer triggers.

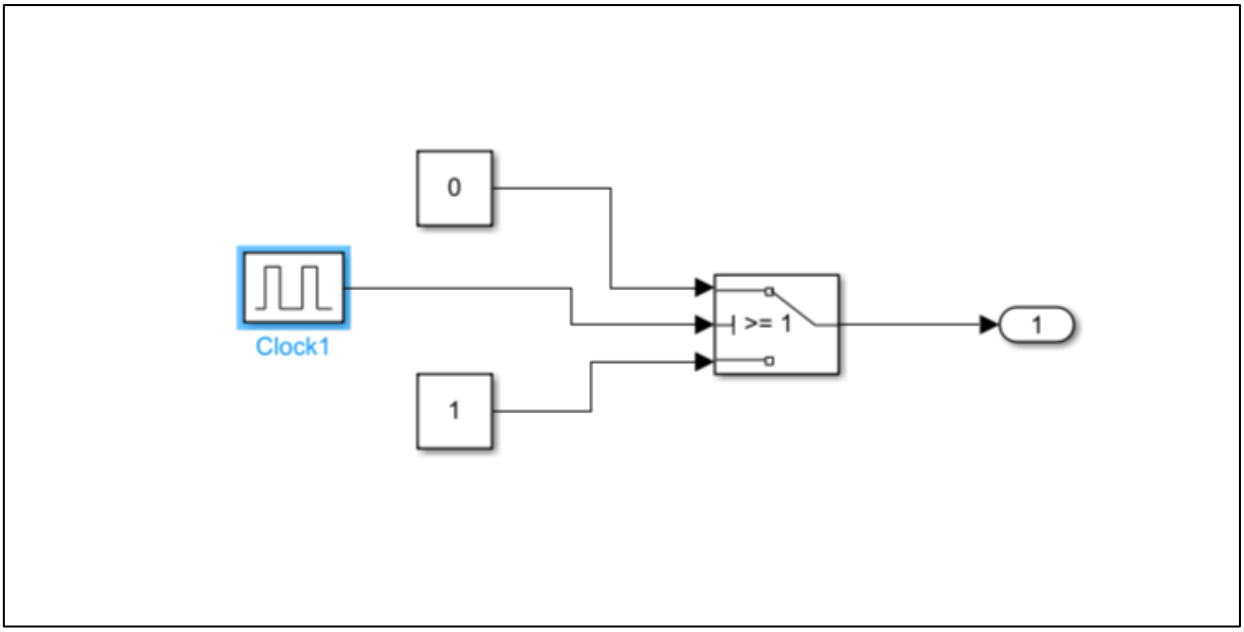

Fig. 9: Input layer of Neuron 1 in Simulink

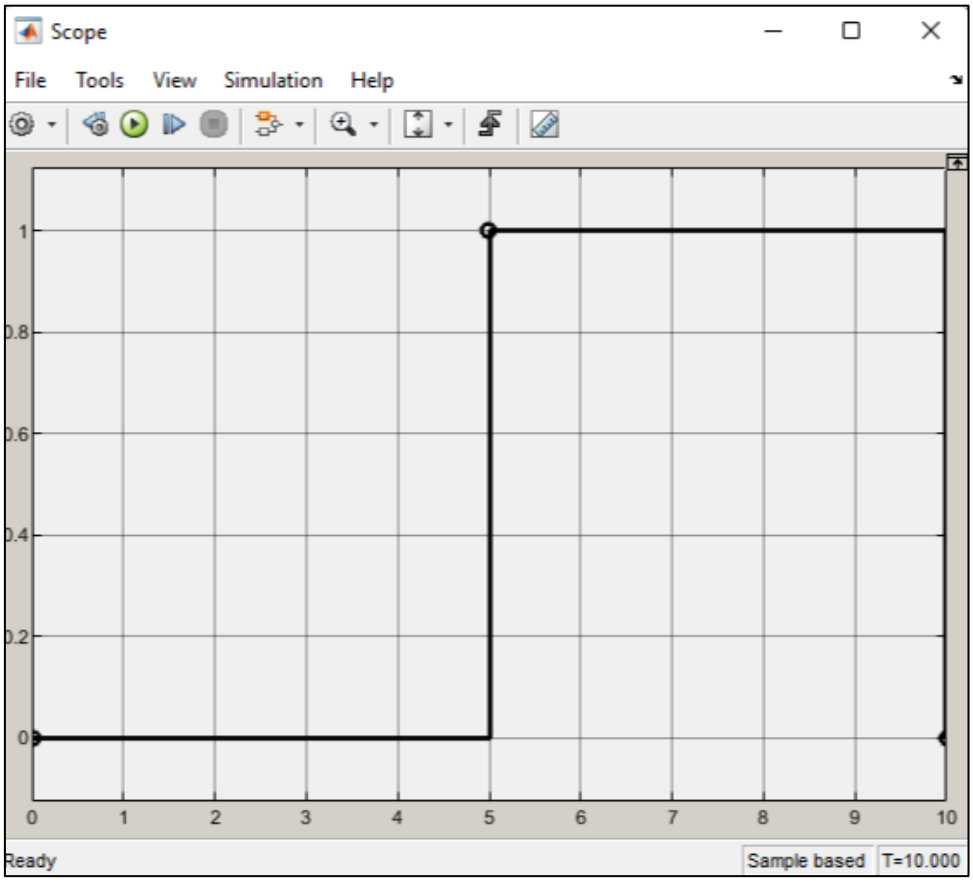

Fig. 10: Output of Input layer of Neuron 1

3.2.2). INPUT LAYER NEURON 2

This input neuron 2 is similar to the previous input neuron except for the time duration of 10 units the output waveform triggers for every 2.5 seconds respectively. Based upon the design shown in fig.11, the corresponding output waveform is obtained and is shown in fig.12.

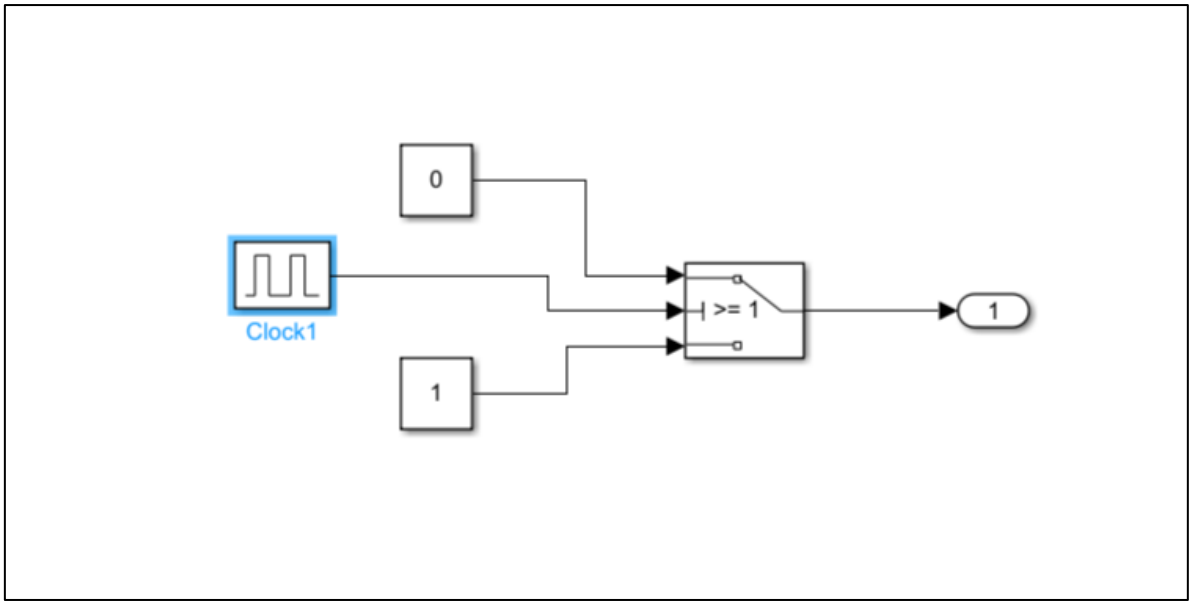

Fig. 11: Input layer of Neuron 2

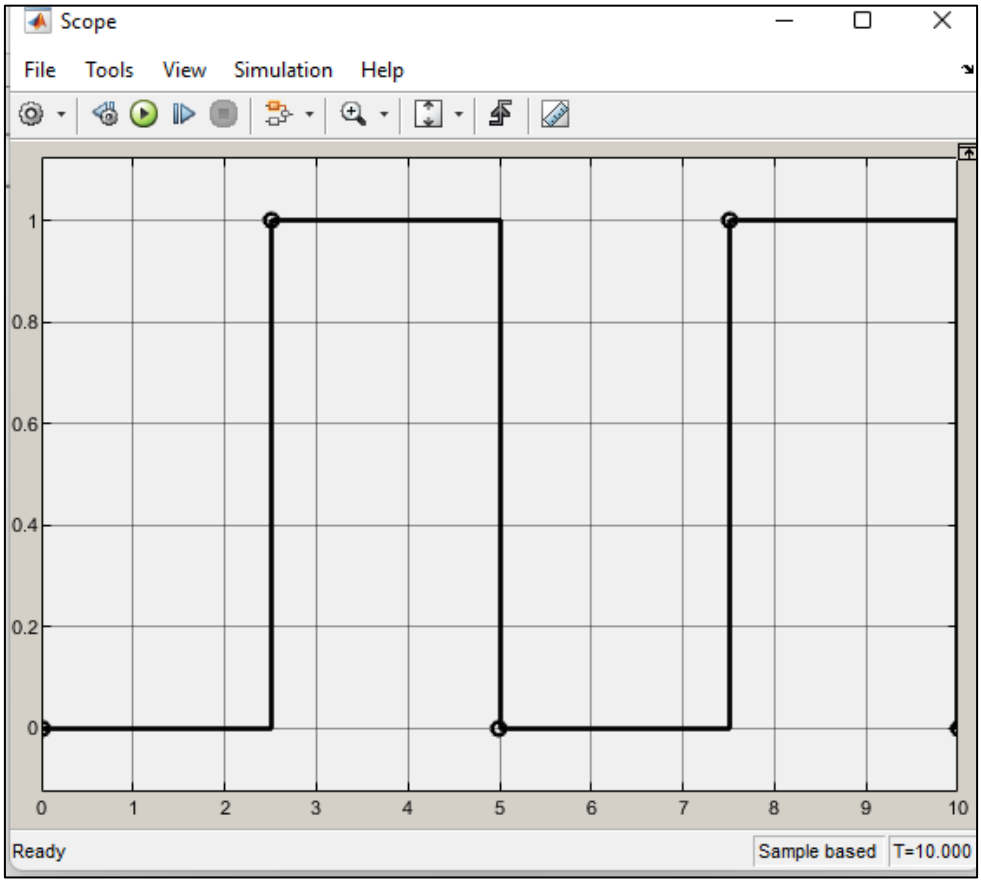

Fig. 12: Output of Input layer of Neuron 2

3.2.3). HIDDEN NEURON 1

A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. It is a typical part of nearly any neural network in which we simulate the activities that occur in the human brain. Hidden neural network layers are set up in many ways. In some cases, weighted inputs are randomly assigned. In other cases, they are fine-tuned and calibrated through a process called back propagation. Based upon the design of hidden layer of neuron 1 as shown in fig. 13, the corresponding output waveform of that neuron is obtained and as shown in fig. 14.

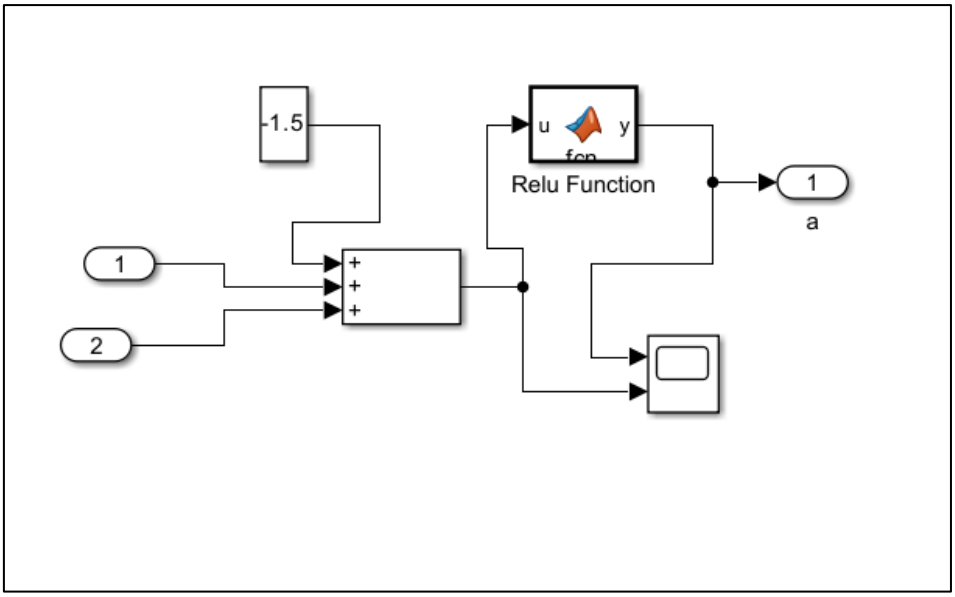

Fig. 13: Hidden layer of Neuron 1

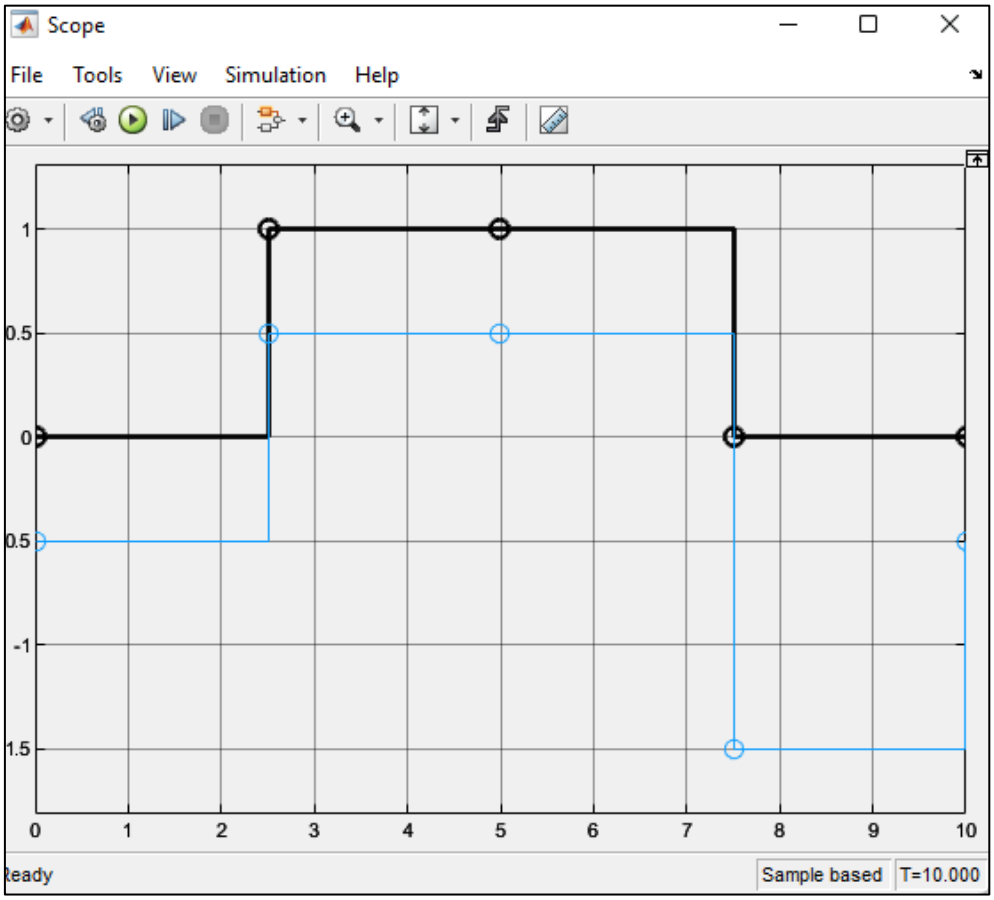

Fig. 14: Output of Hidden layer belongs to neuron 1

3.2.4). HIDDEN NEURON 2

The block diagram of hidden layer 2 is similar to the first hidden layer except the bias input differs. The bias input in the previous hidden layer is -1.5, but here the input bias applied is -0.5. The output waveform of the hidden layer 2 is shown below; the summation output present in the block diagram varies from -0.5 to 1.5. The activation function coverts the negative input as logic 0. And the values above zero are set as high or logic 1 by the activation function. Based upon the design shown in fig.15 the corresponding output is obtained and is as shown in fig. 16.

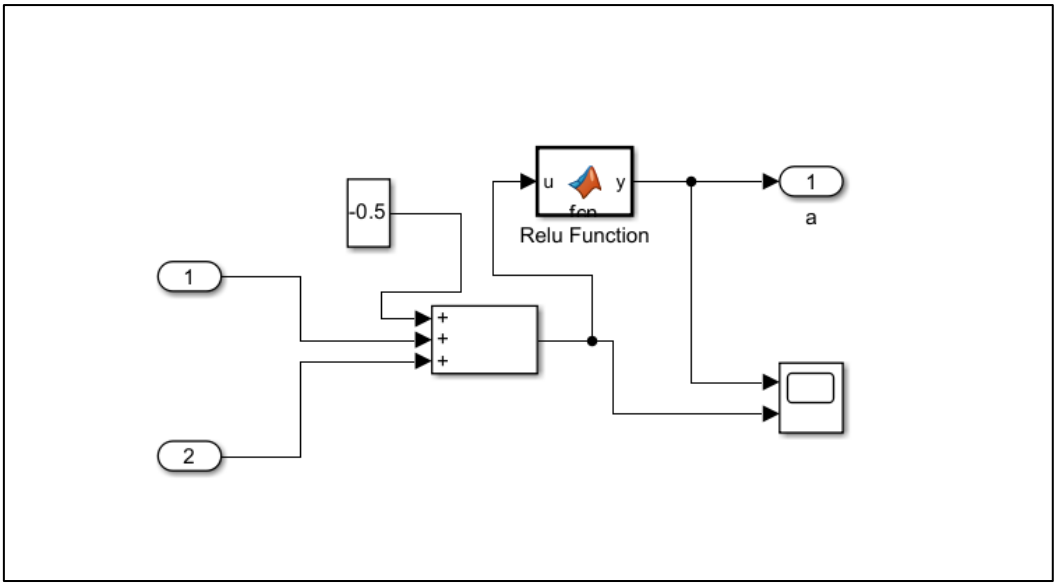

Fig. 15: Hidden layer of Neuron 2

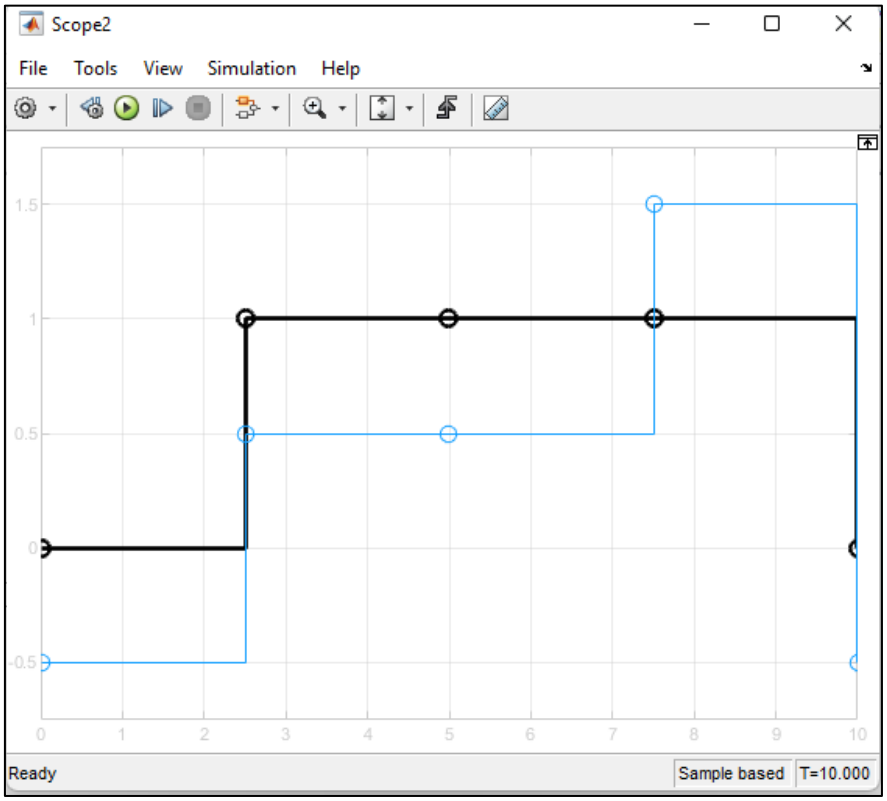

Fig. 16: Output of Hidden layer belongs to neuron 2

3.2.5). ACTIVATION FUNCTION

Here the activation function used is ReLU function. The input signal is denoted as 'u' and the output signal is denoted as 'y'. As you can see from the fig.17, if the input function is greater than or equal to 0, then the output signal becomes high or gives logic '1'. Else if the input function is less than zero then the output is set as low or logic '0'. The piece of code for activation function used in this implementation is as shown below.

```
function y = fcn(u)
if(u>=0)
   y=1;
else
y = 0;
end
```

Fig. 17: Matlab code for ReLU function

The fig.18 represents the output layer which receives input from the two hidden layers. The first hidden layer node is having a weight of -2, while the second hidden layer node is having a weight associated as 1.

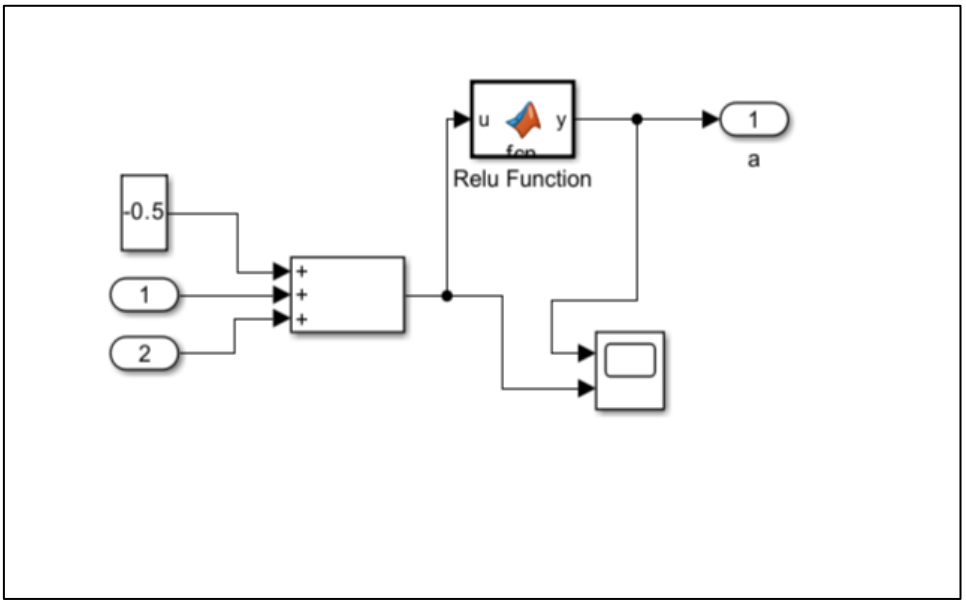

Fig. 18: Neuron of output layer

3.2.6). OUTPUT LAYER

The waveform shown in fig.19 is output waveform obtained from the output layer. The blue line represents the summation output, and the black line represents the activation function.

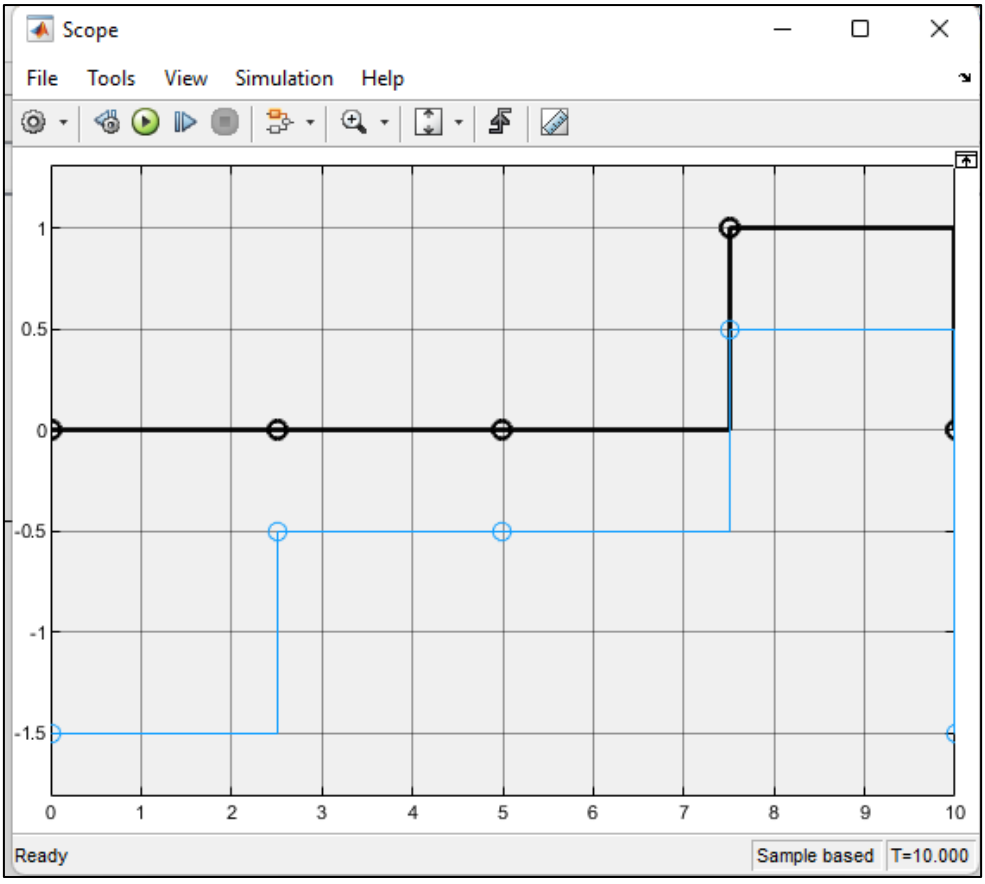

Fig. 19: Output of output neuron

© 2023 | Published by Scholars Middle East Publishers, Dubai, United Arab Emirates

4. COMPARATIVE ANALYSIS

MATLAB SIMULINK	GOOGLE COLAB
Simulink uses Matlab code .	Google colab uses python code.
Easy to build neural network.	Building a neural network is complex.
Simulink has in-built and predefined library blocks to construct the neural network.	Google colab does not have in built and predefined library block to construct the neural network.
No requirement to import additional libraries while coding in Matlab.	There is a requirement to import additional libraries while coding in Python.
Less Flexible	More Flexible
For complex neural networks Matlab is preferable.	For complex neural networks Google colab is less preferable.
It has a dedicated software for 'Deep learning' applications.	For Deep learning applications we use tensorflow and its libraries.

5. CONCLUSION

We can conclude that XOR problem is non-linear system and has got the solution by implementing it by using neural network with one hidden layer. Back propagation algorithm is used to tune synaptic weights in order to get desired results of XOR output. Performance analysis is carried out for XOR gate logic using Matlab and Google Colab. Matlab code is more efficient for analysis of the model and it has dedicated tools for implementing neural network. But Google Colab is widely used as it can implement neural network using one the most widely implemented language i.e., python. Further this can be verified on edge computing hardware platforms like Arduino nano 33 BLE and Raspberry Pi 4.

REFERENCES

1. Piccialli, V., & Sciandrone, M. (2022). Nonlinear optimization and support vector machines. *Ann Oper Res*, 314, 15–47.https://doi.org/10.1007/s10479-022-04655-x
2. Cyr, A., Thériault, F., & Chartier, S. (2020). Revisiting the XOR problem: a neurorobotic implementation. *Neural Comput & Applic*, 32, 9965–9973. https://doi.org/10.1007/s00521-019-04522-0
3. Viswanatha, V., Chandana, R. K., & Ramachandra. A. C. (2022). "Real Time Object Detection System with YOLO and CNN Models: A Review."
4. Borah, P., & Gupta, D. (2020). "Unconstrained convex minimization based implicit Lagrangian twin extreme learning machine for classification (ULTELMC)," *Applied Intelligence*, 50(4), 1327–1344.
5. Behrends, A., & Scheiner, R. (2012). Octopamine improves learning in newly emerged bees but not in old foragers. *J Exp Biol*, 215(7), 1076-1083.
6. Nalepa, J., & Kawulok, M. (2019). Selecting training sets for support vector machines: a review. *Artificial Intelligence Review*, 52(2), 857–900.
7. Viswanatha, V., Chandana, R. K., & Ramachandra, A. C. (2022). IoT Based Smart Mirror Using Raspberry Pi 4 and YOLO Algorithm: A Novel Framework for Interactive Display. *Indian Journal of Science and Technology*, 15(39), 2011-2020.
8. Krichmar, J. L. (2018). Neurorobotics—a thriving community and a promising pathway toward intelligent cognitive robots. *Front Neurorobot*, 12, 42.
9. Yoo, K. H., Koo, Y. D., Ju, H. B., & Na, M. G. (2017). Identification of LOCA and estimation of its break size by multiconnected support vector machines. *IEEE Transactions on Nuclear Science*, 64(10), 1.
10. Zhang, L., Lu, X., & Lu, C. (2017). National matriculation test prediction based on support vector machines. *Journal of University of Science & Technology of China*, 47(1), 1–9,
11. Viswanatha, V., Ashwini, K., & Sathisha, B. M. (2022). Implementation of IoT in Agriculture: A Scientific Approach for Smart Irrigation. 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon). *IEEE*.
12. Tanino, T., Kawachi, R., & Akao, M. (2017). Performance evaluation of multiobjective multiclass support vector machines maximizing geometric margins. *Numerical Algebra Control & Optimization*, 1(1), 151–169.
13. Gu, W., Chen, W.-P., & Ko, C.-H. (2018). Two smooth support vector machines for -insensitive regression. *Computational Optimization & Applications*, 70(1), 1–29.
14. Taherzadeh, G., Zhou, Y., Liew, A. W.-C., & Yang, Y. (2016). Sequence-based prediction of protein-carbohydrate binding sites using support vector machines. *Journal of Chemical Information and Modeling*, 56(10), 2115–2122.
15. Balasundaram, S., & Gupta, D. (2016). Knowledge-based extreme learning machines. *Neural Computing and Applications*, 27(6), 1629–1641.
16. Viswanatha, V., Sathisha, B. M., & Ashwini, K. (2022). Custom Hardware and Software Integration: Bluetooth Based Wireless Thermal Printer for Restaurant and Hospital Management. 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon). *IEEE*.

17. Ehrentraut, C., Ekholm, M., Tanushi, H., Tiedemann, J., & Dalianis, H. (2016). Detecting hospital-acquired infections: a document classification approach using support vector machines and gradient tree boosting. *Health Informatics Journal*, 24(1), 24–42.
18. Zhang, X., Li, Y., & Peng, X. (2016). Brain wave recognition of word imagination based on support vector machines. *Chinese Journal of Aerospace Medicine*, 14(3), 277–281.
19. Viswanatha, V., Ashwini, K., & Pradeep, K. (2021). Internet of things (IoT) based multilevel drunken driving detection and prevention system using Raspberry Pi 3. *International Journal of Internet of Things and Web Services*, 6.

© 2023 | Published by Scholars Middle East Publishers, Dubai, United Arab Emirates