Lab Report on NEURAL NETWORKS (January 2020)

Jagadish Khadka(2073/BEX/313),

Krishna Bahadur Ojha(2073/BEX/317),

Rajad Shakya(2073/BEX/332) ,

***Abstract*—This document is a report on the lab session of Data mining on the study of neural network including some elementary neural networks for representing AND and OR functions, activation functions used in neural networks, backpropagation and construction of neural networks in a spreadsheet as well as by programming. For spreadsheet, Microsoft Excel was used. Additionally, Python programming language was used for training a neural network.**

***Index Terms*—activation function, back propagation, neural network, spreadsheet**

I. INTRODUCTION

NEURAL network is a series of algorithms to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Artificial neural networks are computational models which work similar to the functioning of a human nervous system. There are several kinds of artificial neural networks. These type of networks are implemented based on the mathematical operations and a set of parameters required to determine the output.

A neural network simply consists of neurons (also called nodes). These nodes are connected in some way. Then each neuron holds a number, and each connection holds a weight. These neurons are split between the input, hidden and output layer. In practice, there are many layers and there is not any general best number of layers. Each neuron has an activation a and each neuron that is connected to a new neuron has a weight w. One could multiply activations by weights and get a single neuron in the next layer, from the first weights and activations w_1a_1 all the way to. That is, multiply n number of weights and activations, to get the value of a new neuron. The procedure is the same moving forward in the network of neurons, hence the name feed forward neural network.

II. 'AND' AND 'OR' FUNCTIONS WITH NEURAL NETWORK

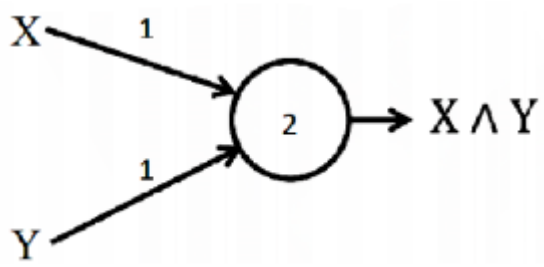
Boolean algebra deals with binary variables and logic operation. A Boolean Function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1, and the logic operation symbols.

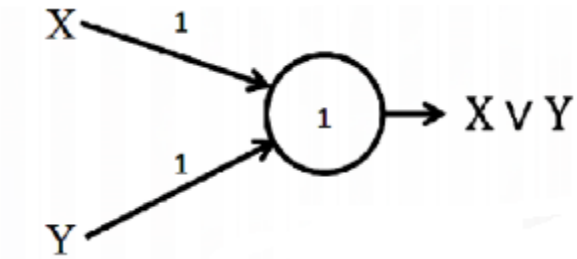
The Logic AND gives the output 1 if and only if all the input events are 1 else 0. The Logic OR gives the output 1 if at least one of the input events is 1 else 0.

The truth table of 'AND' and 'OR' function are shown below:

A	**B**	**A ∧ B**	**A ∨ B**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Neural Network model of the 'AND' and 'OR' Boolean functions are shown below:





Learning factor impacted the number of iterations:
If we increase the learning factor also increase the iteration

III. PYTHON GENERATED ACTIVATION FUNCTIONS USED IN NEURAL NETWORKS

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

If the activation function is linear then the neuron composed only linear functions cannot learn as all the layers will behave same way as composition of two linear function is also a linear function. A non linear activation function will let it learn as per the difference w.r.t error. Hence the activation function must be nonlinear.

Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.

Non-linear function allow back propagation because they have a derivative function which is related to the inputs.

Commonly used activation function are below:

1. Sigmoid or Logistic
2. Tanh - Hyperbolic Tangent
3. Relu – Rectified Linear Units
4. Leaky-Relu

1) SIGMOID OR LOGISTICS

The sigmoid function is a activation function in terms of underlying gate structured in co-relation to Neurons firing, in Neural Networks that creates a flexible S-shaped curve with a minimum value approaching from zero and a maximum value approaching 1

Sigmoid function represented by following formula:

$$\sigma(x) = s = \frac{1}{1 + e^{-x}} \tag{1}$$

And the derivative can be derived as:

$$\sigma'(x) = \lim_{h \to 0} \frac{\sigma(x + h) - \sigma(x)}{h} \tag{2}$$

$$\sigma'(x) = \lim_{h \to 0} \frac{\frac{1}{1 + e^{-x-h}} - \frac{1}{1 + e^{-x}}}{h} \tag{3}$$

$$\sigma'(x) = \lim_{h \to 0} \frac{1 + e^{-x} - 1 - e^{-x-h}}{h(1 + e^{-x-h})(1 + e^{-x})} \tag{4}$$

$$\sigma'(x) = \lim_{h \to 0} \frac{e^{-x}(1 - e^{-h})}{h(1 + e^{-x-h})(1 + e^{-x})} \tag{5}$$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})(1 + e^{-x})} \tag{6}$$

$$\sigma'(x) = \left(\frac{1}{1 + e^{-x}}\right)\left(1 - \frac{1}{1 + e^{-x}}\right) \tag{7}$$

$$\sigma'(x) = s * (1 - s) \tag{8}$$

Normally this function is used in output layer of the binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 and it's derivative are the also function of input so we get more efficient output.

2) TANH – HYPERBOLIC TANGENT

The function produces outputs in scale of [-1, +1]. Moreover, it is continuous function. In other words, function produces output for every x value so this works almost always better than sigmoid function is Tanh function.

Tanh function represented by following formula:

$$f(x) = a = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \tanh(x) \tag{9}$$

And its derivative is:

$$f'(x) = (1 - a^2) \tag{10}$$

It's a non-linear function whose output ranges from -1 to +1. It is usually used in hidden layers of the neural network it concentered along to Centre This makes learning for the next layer much easier.

3) RELU – RECTIFIED LINEAR UNIT

ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as y = max(0, x).this is most commonly used activation function in neural networks

It is defined and denoted as follows:

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases} = \max(0, x) \tag{11}$$

Its derivative is:

$$f'(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined if } x = 0 \end{cases} \tag{12}$$

It is a non-linear function whose value ranges from 0 to infinity. It gives the output x if x is positive and 0 The derivative of the rectified linear function is also easy to calculate. Recall that the derivative of the activation function is required when updating the weights of a node as part of the back propagation of error.

The derivative of the function is the slope. The slope for negative values is 0.0 and the slope for positive values is 1.0.

Traditionally, the field of neural networks has avoided any activation function that was not completely differentiable, perhaps delaying the adoption of the rectified linear function and other piecewise-linear functions. Technically, we cannot calculate the derivative when the input is 0.0, therefore, we can assume it is zero. This is not a problem in practice.
Relu is generally not used in the output layer as its value ranges from zero to infinity which means that it cannot blow up the activation

4) LEAKY RELU FUNCTION

Leaky ReLUs are one attempt to fix the "dying ReLU" problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of 0.01, or so).
The leaky Relu function is defined and denoted as:

$$f(x) = \begin{cases} 0.01x \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases} = \max(0.01x, x) \quad (13)$$

Its derivative is:

$$f'(x) = \begin{cases} 0.01 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases} \quad (14)$$

When 0.01 is replaced by any other quantity then it is called Randomized Relu instead of leaky Relu. While it solves the dying Relu problem, it does not provide consistent predictions for negative input values.

IV. LOSS FUNCTION OR COST FUNCTION

Loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. It is used to determine the error between the output of the algorithm and the given target value. It expresses how far off the mark our computed output is.

Loss functions are used in the optimization problem with the goal of minimizing the loss. Loss functions are used in regression when finding a line of best fit minimizing the overall loss of all the points with the prediction from the line. Loss functions are used while training perceptron and neural networks by influencing how their wights are updated. The larger the loss, larger the update. By minimizing the loss, the model accuracy is maximized

Some of the common loss functions are explained below:
1) MEAN SQUARED ERROR (MSE)

The squaring of the error is self-explanatory, as the error can be positive or negative, so when we would sum the error of all the nodes, the positives and negatives may cancel out. So, to prevent that cancelling effect we square the errors. we can take the absolute value of the error too. but remember Gradient Descent. Which is effect on back propagation

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \quad (21)$$

Where y_i is target value for input to output pair $(\overrightarrow{x_I}, yi)$ and $\hat{y_I}$ is the computed output of the network on input $\overrightarrow{x_I}$.

V. BACKPROPAGATION

Backpropagation, short form of 'Backward Propagation of Errors', is an algorithm for supervised learning algorithm of artificial neural networks using gradient descent. Given an artificial neural network and a loss function, the method calculates the gradient od the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptron to multiplayer feedforward neural networks.

The "backwards" part of the name stems from that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of the weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for previous layer. This backward flow of the error information allows for efficient computation of the gradient at each layer versus the naïve approach of calculating the gradient of each layer separately.

Back propagation is analogous to calculating the delta rule for a multilayer feed forward network. Thus, back propagation require three things; datasets, feed forward neural network and error or loss function.
The derivation of back propagation algorithm is fairly straight forward. It follows from the use of the chain rule and product rule in differential calculus. Application of these rules is dependent on the differentiation of the activation function and the loss function.
Let's consider,

C = Cost function
y = Actual output value
w = weight
a = activation
b = bias
z = w * a + b

Also suppose one hidden layer. Then, the input layer is layer 0, the hidden layer is layer 1 and the output layer is layer 2.So.

$$z^{(L)} = w^{(L)} * a^{(L)} + b \quad (22)$$

$$a^{(L)} = \sigma(z^{(L)}) \quad (23)$$

$$C = (a^{(L)} - y)^2 \quad (24)$$

Now,

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}} = 2(a^{(L)} - y)^2 \sigma'(z^{(L)})a^{(L)} \quad (25)$$

Here, $a^{(L)}$ is an output from the activation function, which

in most of the cases like in sigmoid, Relu, etc. is always positive. Suppose the remaining term in the right-side expression is negative then the gradient would be negative similarly if the remaining term in the right-side expression is positive then the gradient would be positive. Essentially, either all the gradients connected the same neuron in a layer are positive or all the gradients in a layer are negative. This restricts the possible update directions i.e. gradients can move only in the first quadrant and the third quadrant. In other words, most activation functions have outputs in the 1st and 3rd quadrants

Then, for updating the weights, biases and activation:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} \quad (26)$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} \quad (27)$$

$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \quad (28)$$

The gradient is computed according to a mini-batch of the data. For each observation in the mini-batch, the output for each weight and bias is averaged. Then the average of those weights and biases becomes the output of the gradient which creates a step in the average best direction over the mini-batch size.

Then the weights and biases are updated after each mini-batch. Each weight and bias are changed by a certain amount for each layer L:

$$w^{(L)} = w^{(L)} - \eta * \frac{\partial C}{\partial w^{(L)}} \quad (29)$$

$$b^{(L)} = b^{(L)} - \eta * \frac{\partial C}{\partial b^{(L)}} \quad (30)$$

Where, η is the learning rate.

Updating the weights and biases in layer 2 (or L) depends only on the cost function, and the weights and biases connected to layer 2. Similarly, for updating layer 1 (or L−1), the dependencies are on the calculations in layer 2 and the weights and biases in layer 1. This would add up, if there are more layers, there would be more dependencies.

So, for layer 2,

$$\frac{\partial C}{\partial w^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}} \quad (31)$$

$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial b^{(2)}} \quad (32)$$

Then, for layer 1,

$$\frac{\partial C}{\partial w^{(1)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}} \quad (33)$$

$$\frac{\partial C}{\partial b^{(1)}} = \frac{\partial C}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b^{(1)}} \quad (34)$$

VI. METHODOLOGY

A. LIBRARY USED

1) Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy which produces publication quality figures in variety of hardcopy formats and interactive environment across platforms. Matplotlib can be used in python scripts in python and Ipython shells, Jupyter Notebook, web application and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

2) Numpy

It is a fundamental package for scientific computing with python. It is used as a multidimensional container of generic data.it is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

B. NEURAL NETWORK FOR AND – OR FUNCTION

The Perceptron algorithm states that:

$$\text{Prediction } (y`) = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases} \quad (1)$$

Where, Wx is the sum of the products of weights and inputs.

Also, the steps in this method are similar to how Neural Networks learn, which is as follows;

1. Initialize weight values and bias
2. Forward Propagate
3. Check the error
4. Backpropagate and Adjust weights and bias
5. Repeat for all training examples

Since the output of an AND gate is 1 only if both inputs (in this case, x1 and x2) are 1. So, following the steps listed above;

1) Row 1

- From w1*x1+w2*x2+b, initializing w1, w2, as 1 and b as 1, we get, x1(1) +x2(1) +1
- Passing the first row of the AND logic table (x1=0, x2=0), we get, 0+0+1 = 1

- From the Perceptron rule, if Wx+b >=0, then y`=1. This row is incorrect, as the output is 0 for the AND gate.
- So, the bias was changed according to equation AHSJHASH to 0.5, taking learning rate as 0.5, and the weights remained unchanged as both x1 and x2 are 0.

2) Row 2

- Passing (x1=0 and x2=1), we get, 0+1+0.5 = 1.5
- Similar to the above condition, this row is incorrect and following equation AHSJAJSH the bias was changed to 0 and w1 remained unchanged while w2 was changed to 0.5.

3) Row 3

- Passing (x1=1 and x2=0), we get, 1+0+0 = 1
- Similarly, as the row is incorrect, bias was changed to -0.5, w1 to 0.5 and w2 remained unchanged as 0.5.

4) Row 4

- Passing (x1=1 and x2=1), we get, 0.5+0.5–0.5 = 0.5
- From the perceptron rule, this is valid as the actual output in this case is 1, so no back propagation was required.

This process was iterated until all the errors for the four rows became zero (0), i.e. no error was found.

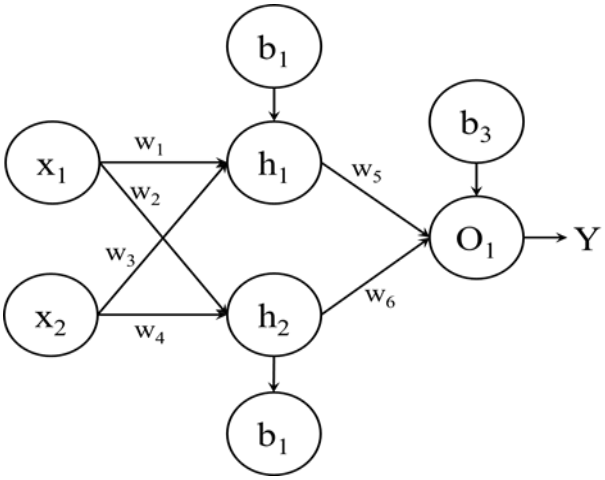
From this, the values of weights and biases were determined. This process was done a few more times by changing the learning rate to compare the different number of iterations required.

Similarly, the same process was done for OR function with the only exception of the desired outputs for the four rows to be 0,1,1,1 respectively

C. NEURAL NETWORK PROGRAMMED WITH PYTHON

1) Neural Network Classifier Used

The following neural network was built using python,



Here,

$$y = \sigma(b_3 + w_6 h_2 + w_5 h_1) = \sigma(s) \quad (35)$$

$$h_2 = \sigma(b_2 + w_4 x_2 + w_3 x_1) = \sigma(t) \quad (36)$$

$$h_1 = \sigma(b_1 + w_2 x_2 + w_1 x_1) = \sigma(u) \quad (37)$$

$$C = \frac{1}{2}(y - \hat{y})^2 \quad (38)$$

$$\frac{\partial C}{\partial y} = y - \hat{y} \quad (39)$$

So,

$$\frac{\partial C}{\partial w_6} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial w_6} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)h_2 \quad (26)$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial w_5} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)h_1 \quad (27)$$

$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial b_3} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big) \quad (28)$$

$$\frac{\partial C}{\partial w_4} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_2}\frac{\partial h_2}{\partial t}\frac{\partial t}{\partial w_4} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(t)\big(1 - \sigma(t)\big)w_6 x_2 \quad (26)$$

$$\frac{\partial C}{\partial w_3} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_2}\frac{\partial h_2}{\partial t}\frac{\partial t}{\partial w_3} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(t)\big(1 - \sigma(t)\big)w_6 x_1 \quad (27)$$

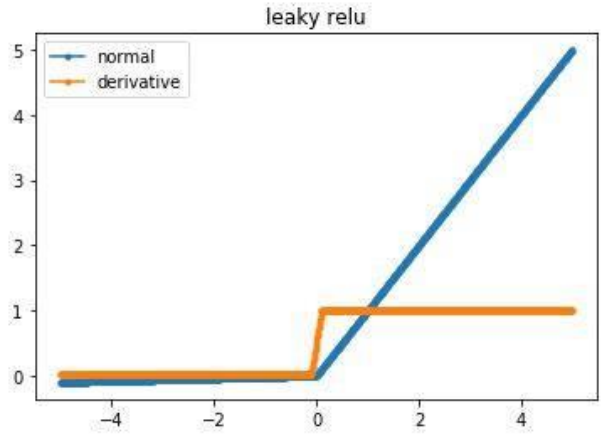
$$\frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_2}\frac{\partial h_2}{\partial t}\frac{\partial t}{\partial b_2} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(t)\big(1 - \sigma(t)\big)w_6 \quad (27)$$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_1}\frac{\partial h_1}{\partial u}\frac{\partial u}{\partial w_2} = (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(u)\big(1 - \sigma(u)\big)w_5 x_2 \quad (28)$$

(26)

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_1}\frac{\partial h_1}{\partial u}\frac{\partial u}{\partial w_1}$$
$$= (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(u)\big(1 - \sigma(u)\big)w_5 x_1$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial h_1}\frac{\partial h_1}{\partial u}\frac{\partial u}{\partial b_1} \quad (28)$$
$$= (y - \hat{y})\sigma(s)\big(1 - \sigma(s)\big)\sigma(u)\big(1 - \sigma(u)\big)w_5$$



VII. RESULT

A. NEURAL NETWORK FOR AND - OR FUNCTION

The table below shows the results of weights obtained while using different learning factors for AND and OR functions:

Function	η	Iterations Required	Final Values Obtained		
			Bias	Weight 1	Weight 2
AND	0.5	6	-1.5	1	0.5
	0.05	12	-0.45	0.45	0.45
	0.02	29	-0.46	0.44	0.44
	0.01	57	-0.45	0.45	0.44
OR	0.5	3	-0.5	1	1
	0.05	21	~0	1	1
	0.02	51	~0	1	1
	0.01	101	~0	1	1

From the table, it can be cleared that decreasing the learning rate, increases the number of iterations during training period and the weight values were obtained much lower for lower learning rate. Thus, lower the learning rate higher is the training period.

Hence, the OR function can be written as:

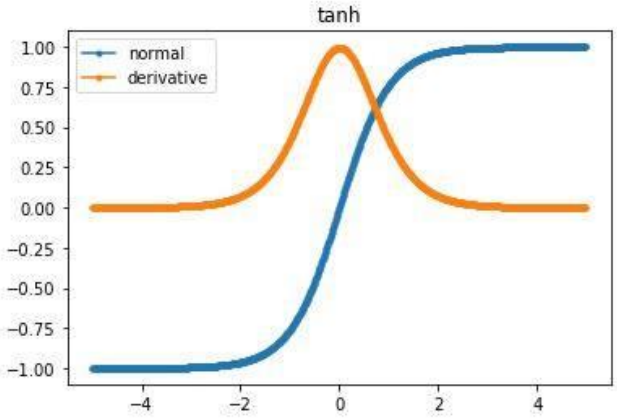
Z = x1 + x2

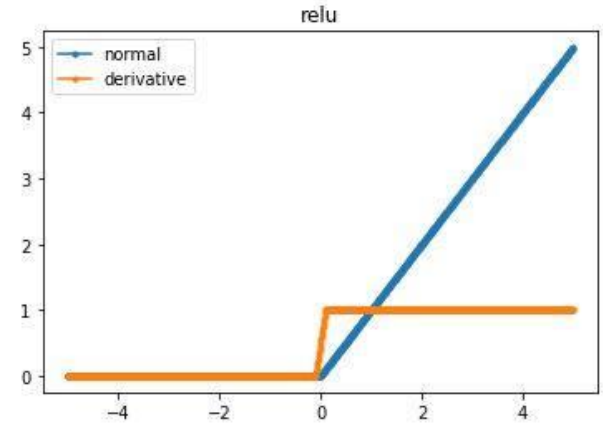
And, the AND function can be written as:

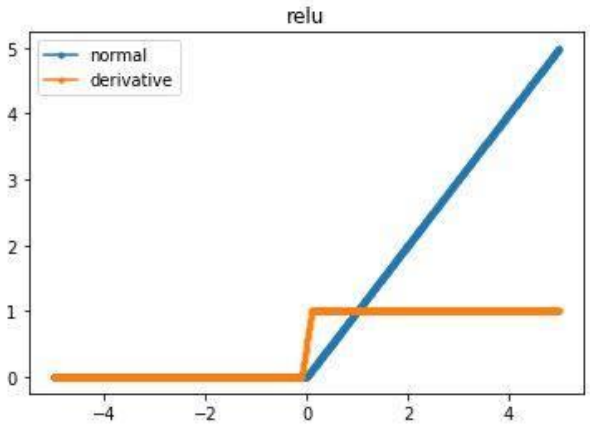
Z = 0.45x1 + 0.45x2 + 0.45

B. ACTIVATION FUNCTIONS

The graph of the activation functions with their derivative are shown below:





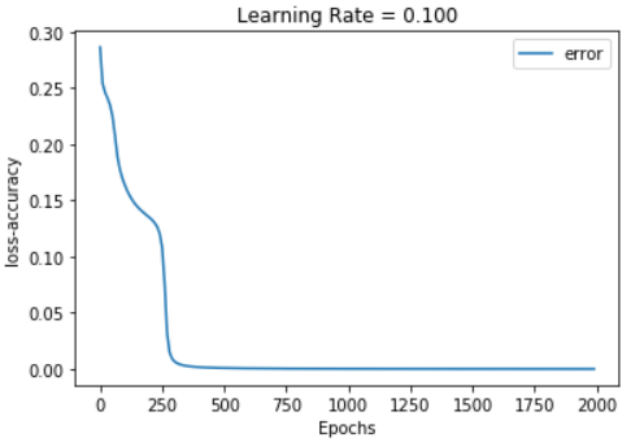


C. NEURAL NETWORK PROGRAMMED USING PYTHON

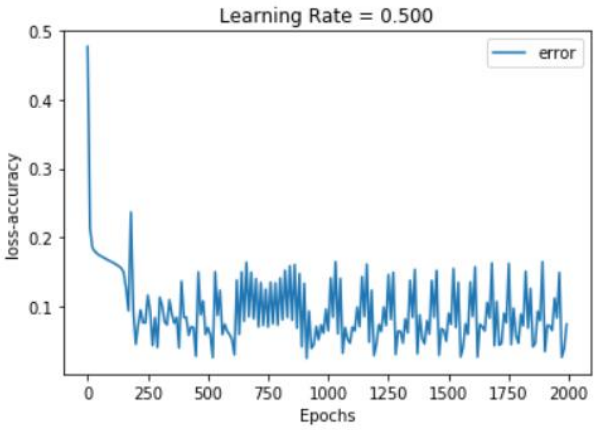
Neural network was programmed using python programming language. We trained neural network on different activations functions for hidden and output layer. According to their usage we tried to compare their performance based on error vs epoch curves, loss and accuracy.

We experimented in four cases where we vary the activations functions on hidden and output layer according to following cases. Below is presented the cases for learning rate 0.1

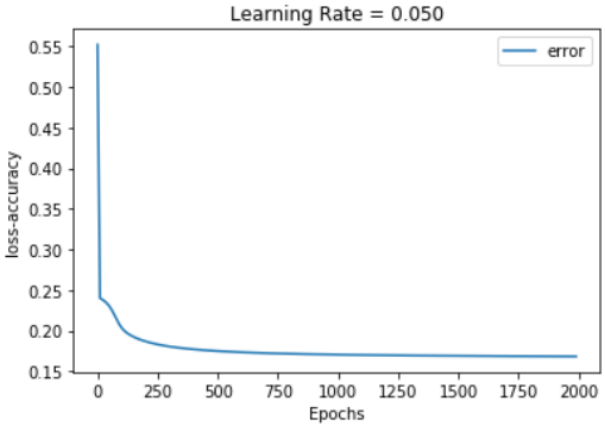
1) Tanh activation functions in hidden layers and output layer



On training the neural network on learning rate 0.1 on the 1999 epoch loss was found to be 0.0 and accuracy of 0.933 was observed.

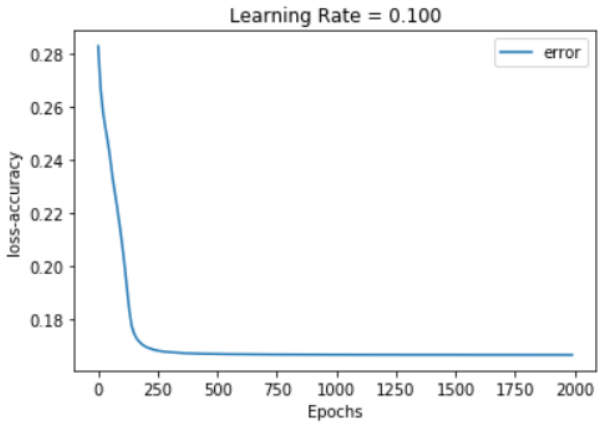


On training the neural network on learning rate 0.5 on the 1999 epoch loss was found to be 0.074 and accuracy of 0.823 was observed.

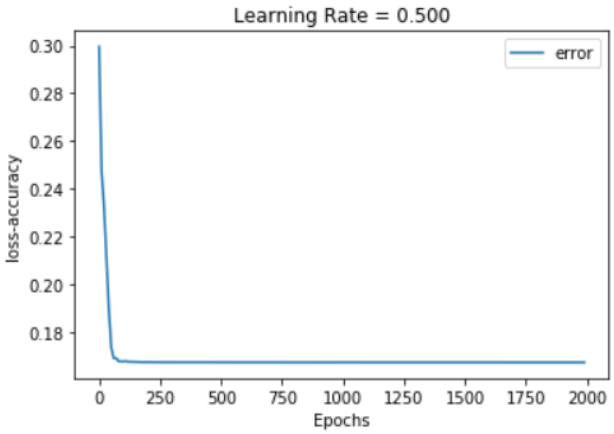


On training the neural network on learning rate 0.05 on the 1999 epoch loss was found to be 0.168 and accuracy of 0.663 was observed.

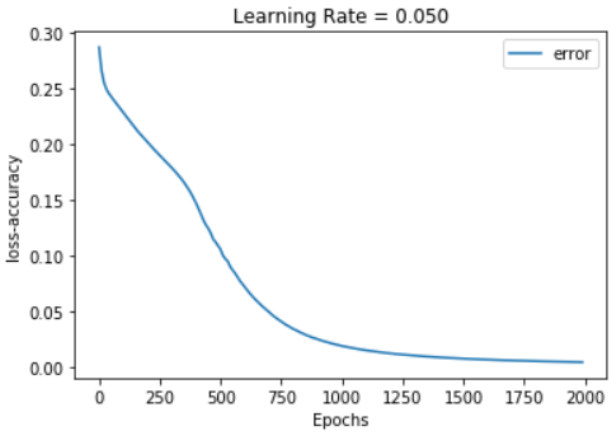
2) ReLU activation function in hidden layer and Sigmoid in output layer



On training the neural network on learning rate 0.1 on the 1999 epoch loss was found to be 0.167 and accuracy of 0.663 was observed.

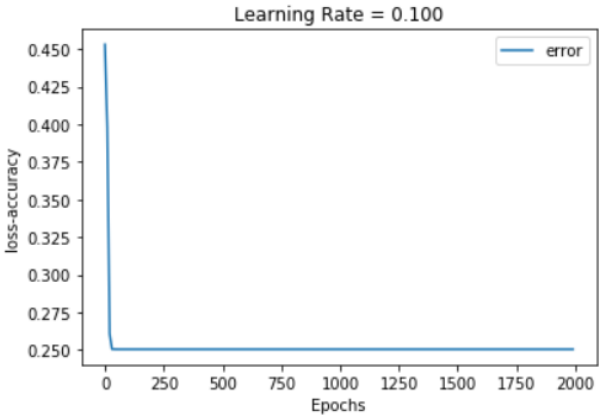


On training the neural network on learning rate 0.5 on the 1999 epoch loss was found to be 0.167 and accuracy of 0.671 was observed.

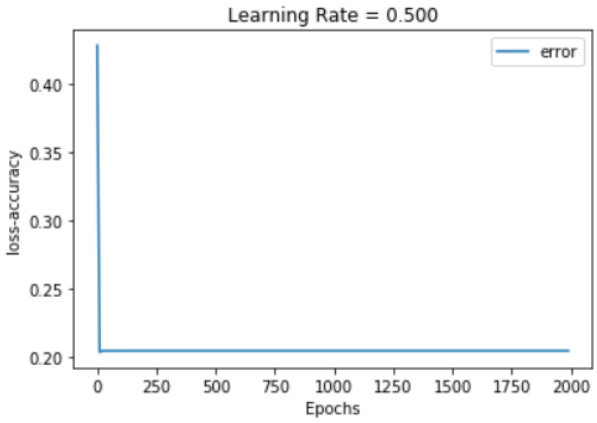


On training the neural network on learning rate 0.05 on the 1999 epoch loss was found to be 0.004 and accuracy of 0.938 was observed.

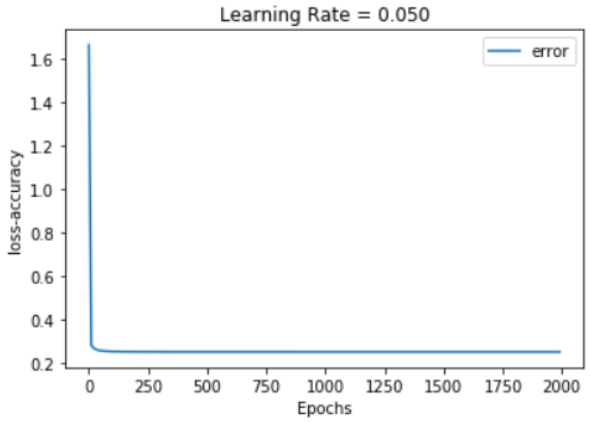
3) ReLU activation function in hidden layer and Tanh in output layer



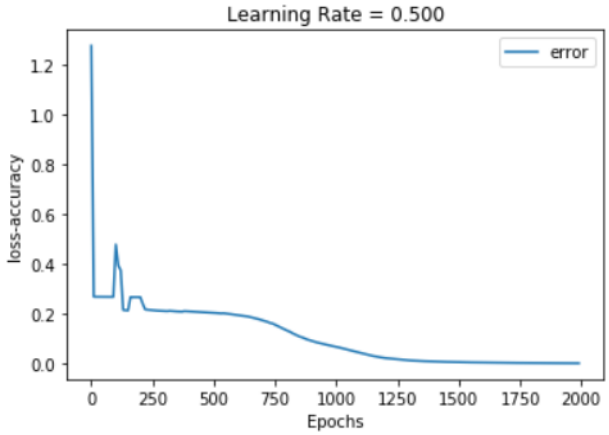
On training the neural network on learning rate 0.1 on the 1999 epoch loss was found to be 0.250 and accuracy of 0.500 was observed.



On training the neural network on learning rate 0.5 on the 1999 epoch loss was found to be 0.204 and accuracy of 0.610 was observed.

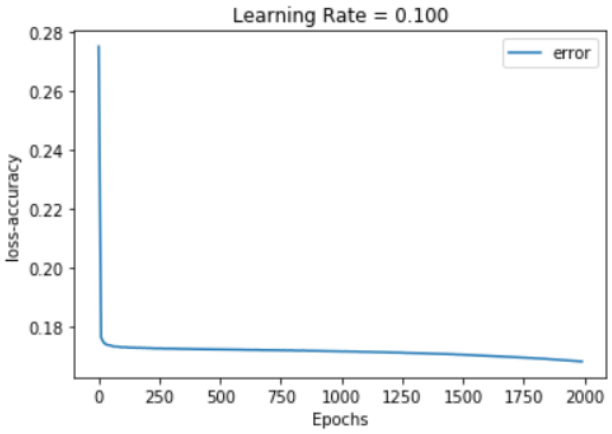


On training the neural network on learning rate 0.05 on the 1999 epoch loss was found to be 0.250 and accuracy of 0.743 was observed.

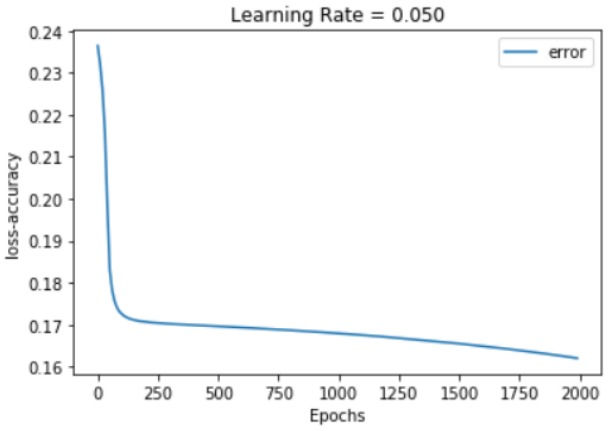


On training the neural network on learning rate 0.5 on the 1999 epoch loss was found to be 0.003 and accuracy of 0.968 was observed.

4) Leaky ReLU (alpha = 0.01) in hidden layer with Tanh in output layer



On training the neural network on learning rate 0.1 on the 1999 epoch loss was found to be 0.168 and accuracy of 0.649 was observed.



On training the neural network on learning rate 0.05 on the 1999 epoch loss was found to be 0.162 and accuracy of 0.656 was observed.

We found that the case where tanh activation used in both hidden and output layer reach the optimum weights more quickly.

VIII. DISCUSSION

From this lab we observed the different activation function used and accuracy, loss, and error vs epochs curves in neural network model

Lab for neural network for AND, OR gate was performed in excel and experimented with different learning rates.

IX. CONCLUSION

We were able to draw conclusions from both python programming and excel implementation of neural network that decreasing learning rate results the increase in iteration in learning.

X. APPENDIX

https://github.com/krishnabojha/Data_mining/tree/master/4.%20Nural%20Network

REFERENCES

[1] Emil M Petriu, Professor, University of Ottawa, "Neural Networks: Basics"
[2] Carlos Gershenson, "Artificial Neural Networks for Beginners", arxiv.org