Neural Representation of Logical Gates

Setareh Roshan
Department of Computer Engineering
Shahid Rajaee Teacher Training University
Tehran, Iran
setare.roshan1996@gmail.com

***Abstract*—Perceptron is an artificial neuron which its algorithm learns the weights for the input signals in order to draw a linear decision boundary. The purpose of this article is to mathematically explain how perceptron updates the weights using logical gates, and it will explain the logic behind how the values are being changed in simple terms. This simple artificial neuron found suitable weights for AND, OR, NAND, and NOR gates, but it didn't work for XOR and XNOR gates.**

Keywords—logical gates, perceptron, machine learning, artificial neuron

I. Introduction

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. Perceptron is an algorithm for learning a binary classifier called a threshold function [1]. A logic gate is a device that acts as a building block for digital circuits. They perform basic logical functions that are fundamental to digital circuits.

Single-layer perceptron is only capable of learning linearly separable patterns. For a classification task with some step activation function, a single node will have a single line dividing the data points forming the patterns. In this article, the perceptron will learn the NAND, AND, OR, and NOR gates, also I will show that perceptron is unable to learn XOR, and XNOR gates.

II. Methods

A. Logical Gates

logic gates will make decisions based on a combination of digital signals coming from its inputs. Most logic gates have two inputs and one output. Logic gates are based on Boolean algebra. At any given moment, every terminal is in one of the two binary conditions, false or true. False represents 0, and true represents 1. Depending on the type of logic gate being used and the combination of inputs, the binary output will differ. I will discuss some of them that I used in this article as follows.

1) AND Gate

The gate acts in the same way as the logical and operator. the output is 1 only when both inputs are 1 (Eq. (1) and (2)

Table IA).

$$input_1 . input_2 = output$$

(1)

2) OR Gate

The output is true if either or both of the inputs are true. If both inputs are false, then the output is false. In other words, for the output to be 1, at least an input has to be 1 (Eq. (2), and Table IB).

$$input_1 + input_2 = output$$

(2)

Table I. shows the logical combination of AND (A), and OR (B) gates.

A			**B**		
Input		***Output***	***Input***		***Output***
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

3) NAND Gate

The NAND gate operates as an AND gate followed by a NOT gate. The output is false, or 0, if both inputs are true, or 1. Otherwise, the output is true, or 1 (Eq. (3), and Table IIA).

$$\overline{input_1 . input_2} = output$$

(3)

4) NOR Gate

The NOR gate is a combination OR gate followed by an inverter. Its output is true, or 1 if both inputs are false, or 0. Otherwise, the output is false, or 0 (Eq. (4), and Table IIB).

$$\overline{input_1 + input_2} = output$$

(4)

Table II. shows the logical combination of NAND (A), and NOR (B) gates.

A			**B**		
Input		***Output***	***Input***		***Output***
0	0	1	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0

5) XOR Gate

The output is true, or 1, if either, but not both, of the inputs are true, or 1. The output is false, or 0, if both inputs are false or if both inputs are true (Eq. (5), and Table IIIA).

$$input_1 . \overline{input_2} + \overline{input_1} = output$$

(5)

6) XNOR Gate

The XNOR gate is a combination XOR gate followed by an inverter. Its output is true if the inputs are the same, and "false if the inputs are different (Eq. (6), and Table IIIB).

$$input_1 . input_2 + \overline{input_1} . \overline{input_2} = output$$

(6)

Table III. shows the logical combination of XOR(A), and XNOR (B) gates.

A			**B**		
Input		***Output***	***Input***		***Output***
0	0	0	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1

B. *Perceptron Learning Rule*

Perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons (an artificial neuron) [2]. A perceptron does certain computations to detect features or business intelligence in the input data. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time. Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not (Fig. 1).

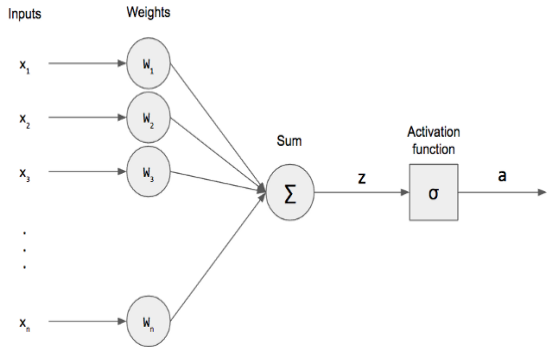

Fig. 1 represents an artificial neuron which x_i inputs are Dendrites, weights are Synapses and output (a) is Axon.

Perceptron is a function that maps its input (x), which is multiplied with the learned weight coefficient; an output value $y(n)$ is generated (Fig. 1). In this study, I used step function as an activation function which generates output as follows:

$$y_t = \begin{cases} 1 & if\ \vec{x}.\vec{w} + b > 0 \\ 0 & otherwise \end{cases} \quad , \quad y_t \in \{0,1\}$$

(7)

In Eq. (7) $\vec{w}$ is vector weights (initial weights are random), and $\vec{x}$ is vector of inputs. Therefore, $\vec{x}.\vec{w}$ is calculates as follows:

$$\sum_{i=1}^{m} w_i . x_i$$

(8)

I modeled logical gates by perceptron. In this modeling, there was a desired output which is equal to Gate's output, also there was an actual output that is calculated using Eq. (7). In next step we will calculate an error (Eq. (9)) by using this error I will be able to update the weights (using Eq. (10)), and lead the error to zero.

$$\overrightarrow{\varepsilon_t} = \overrightarrow{d_t} - \overrightarrow{y_t} , \quad \varepsilon_t \in \{-1,0,+1\}$$

(9)

$$\overrightarrow{w_{t+1}} = \overrightarrow{w_t} + \eta \overrightarrow{\varepsilon_t} \overrightarrow{x_t}$$

(10)

III. Results

I modeled four gates (AND, NAND, OR, and NOR) which they were linearly separable (Fig. 2), so perceptron is guaranteed to converge [3]. I modeled two more gates (XOR, and XNOR) that they aren't linearly separable (Fig. 3), and perceptron is unable to learn them.

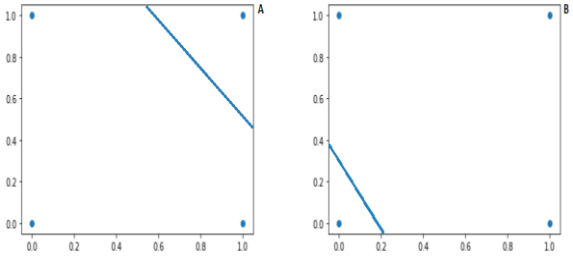

Fig. 2 represents perceptron which produces a line that divide the space into two areas. A. refers to the AND, and NAND gates. B. refers to OR, and NOR gates.

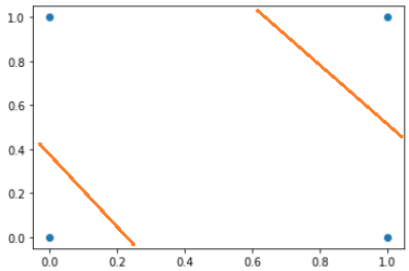

Fig. 3 as you can see these problems (XOR, and XNOR) are not linearly separable.

I also changed the learning rate in only NAND gate from 0.00001 to 0.1 to compare number of epochs necessary for learning, as it's shown in Fig. 1Fig. 4, learning rate 0.00001 isn't suitable at all for learning. As you can see in Fig. 4, learning was too slow, and as the learning rate increased, learning process became faster.

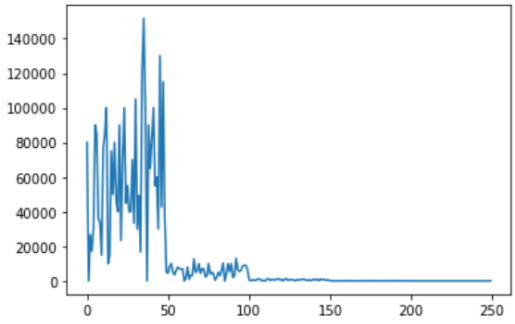

Fig. 4. In this diagram, I compared different amounts of learning rate and how it impacts on the number of epochs that perceptron needs to find the suitable weights. First 50 data on the diagram were given using 0.00001 learning rate, next 50 refers to 0.0001, 100-150 refers to 0.001, 150-200 form 0.01, finally 200-250 is for 0.1.

IV. Conclusion

In conclusion, we can see that perceptron is only capable to learn and solve problems that are linearly separable. In these kinds of problems learning rate plays an important role on the duration of learning process. We understand that learning rates of 0.001 to 0.1 are suitable for this problem. In future works we will show that Multi-layer perceptron is able to solve XOR and XNOR problems. In Table IV, there's some results from implemented perceptron.

Table IV shows initial weights and the final weights of different gates. In these results learning rate was considered 0.01.

AND GATE	Initial	[0.6, 0., 0.1]
	Final	[0.28, 0.02, -0.29]
NAND GATE	Initial	[0.6, 0.4, 0.6]
	Final	[-0.02, -0.09, 0.11]
OR GATE	Initial	[0.8, -0.3, 0.]
	Final	[0.8, 0.01, -0.01]
NOR GATE	Initial	[0.3, 0.2, 0.3]
	Final	[-0.01, -0.03, 0.01]

V. References

[1] Freund, Y.; Schapire, R. E., "Large margine classification using perceptron algorithm," pp. 277-296, 1999.

[2] A. B. Novikoff, "On convergence proofs on perceptrons. Symposium on the Mathematical Theory of Automata," pp. 12, 615-622, 1962.

[3] M. Collins, "Discriminative training methods for hidden Markov models: Theory and experiments with the perceptron algorithm," *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002.