Digital Building Blocks using Perceptron in Neural Networks

Shilpa Mehta

Presidency University Bangalore, India

Corresponding author: Shilpa Mehta, Email: shilpamehta1.official@gmail.com

Most microprocessors and micro-controllers are based on Digital Electronics building Blocks. Digital Electronics gives us a number of combinational and sequential circuits for various arithmetic and logical operations. These include Adders, Subtracters, Encoders, Decoders, Multiplexers, DE multiplexers and Flip Flops. These further combine into higher configurations to perform advanced operations. These operations are done using logic circuits in digital electronics. But in this paper, we explore the human reasoning approach using artificial neural networks. We will look into neural implementations of logic gates implemented with SLP (Single layer perceptron) and MLP (Multi-Layer Perceptron). We will also look into recurrent neural architectures to make basic memory elements, viz. Flip Flops which use feedback and may involve in one or more neuron layers.

Keywords: ANN, Adders, Subtracters, Combinational Circuits, SLP, MLP, Perceptron

1 Introduction

Traditional Digital Electronics works on Boolean logic. Boolean Logic is based on human reasoning and it has only two valid states – TRUE (also called 1/ present / high / on state) and FALSE (also called 0 / absent / low / off state). It uses Basic gates (AND OR NOT) and Universal (NAND NOR) Gates and advanced gates (XOR XNOR). Combinations of these gates are used to implement various digital circuits like Half Adders and Full Adders and also similar subtracters [1]. A full adder is used in single place addition of two binary numbers. A very common IC chip for 4-bit binary numbers' addition is IC 7483 [2]. It has 4 full adders where each place generated carry becomes the input to the next place, it also has a pin for Carry In, and another pin for Carry out. Hence two 7483 ICs may be connected in tandem to get a new circuit with 8 bit addition facility.

Neural networks may provide us with logical implementations of logic gates by being designed for multiple operations at a time. Besides, in traditional logic gate circuits with diodes and transistors, there is no "thinking" behind any operation as in human behaviors, it is just a circuit at work to generate an output bit corresponding to any input combination. In this paper, we will devise ANN for a more "thought" oriented approach to digital operations.

Further operations in this computer age require memory. Flip flops re the basic building blocks of memories [3], [4], and they can remember a "1" or a "0" as per the user's requirements.

Some work has been done on ANN implementations of Boolean combinations, and this paper looks at specific gate implementations only. Applications of Such classes using Fuzzy logic gates for classifying intercepted mail communication is found in [5]. For the circuits in this paper, the programming was done on python and the networks were found to work as expected.

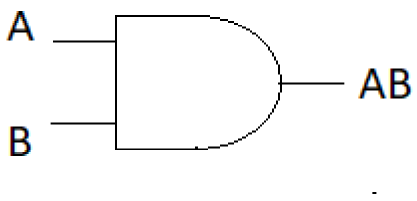
2 Methodology for the Implementation of Basic Gates

Neural Networks for Linearly separable problems work on single layer feed forward architectures. Simple basic gates, viz. AND, OR, NAND, NOR with any number of Inputs can be implemented with such simple architectures. Multiple implementations of Logic gates are already in existence, here we talk of the methodology rather than the actual implementation. [6, 7].

2.1 AND Gate and NAND Gate

The concept of the logic gate AND is that it may have multiple inputs, but the output will be able to attain a TRUE state if and only if all the "m" inputs in the problem of consideration are true simultaneously. It is a perfectly linearly separable problem in the m dimensional input space. It resembles the Intersection of two sets.

When we talk of a “Logic Gate” we should be talking a logic-oriented approach to the design of the neuron. The logic of the AND gate is that in the absence of any of the inputs being active, the output should be false. This obviously implies that the bias should be negative if we take a threshold activation function. The output should stay false for any “m-1” inputs taken together, but when all “m” inputs get active, the output takes a “True” state. That means the combination on ALL the inputs acting together should be able to “defeat” the bias and take the output to a “True” state while no other combination should succeed in this. Hence the weights of all inputs may be taken as (+1) while he bias can be taken as a negative value less than the number of inputs (m) but greater than (m-1).



A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 1. The symbol and Truth table of a 2 input AND Gate

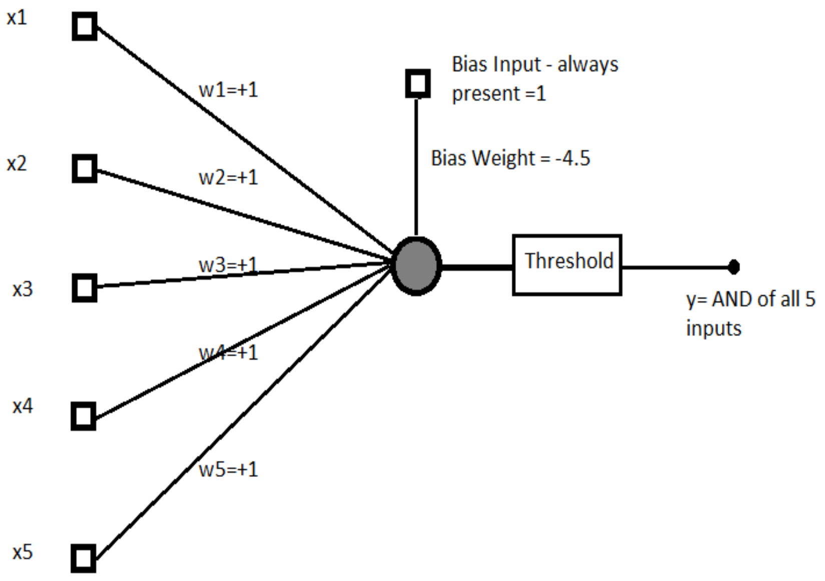


Figure 2. AND Gate with 5 inputs

Figure 2 shows the implementation of a 5 Input AND GATE the bias is chosen as (-4.5) for a gate with 10 inputs the bias weight may be (-9.5) when all inputs have a weight of +1. In case we have more than 10 inputs – all the weights may be scaled down by a factor of ten. For example, for 100 input AND gate in place of keeping bias as (-99.5) and weights as (+1); it may be scaled down to (-9.95) and all weights may be (+0.1)

As we know – NAND gate is a universal gate and it operates exactly opposite to AND Gate. When all the inputs are in true state then the NAND output goes false, for all other combinations the output stays "True". Hence, we can simply design AND gate and then invert all polarities for all inputs including the bias. Another approach could be to keep weights unchanged but invert the activation function.

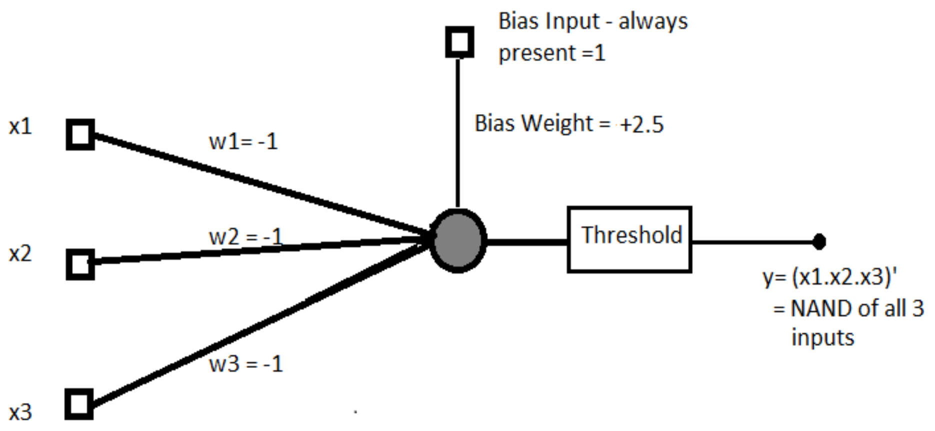
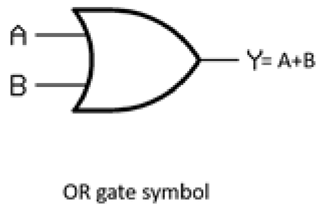

Bias Input - always present =1
Bias Weight = +2.5
x1
x2
x3
w1= -1
w2 = -1
w3 = -1
Threshold
y= (x1.x2.x3)' = NAND of all 3 inputs

Figure 3. A 3 inputs NAND Gate Implementation with SLP

2.2 OR Gate and NOR Gate

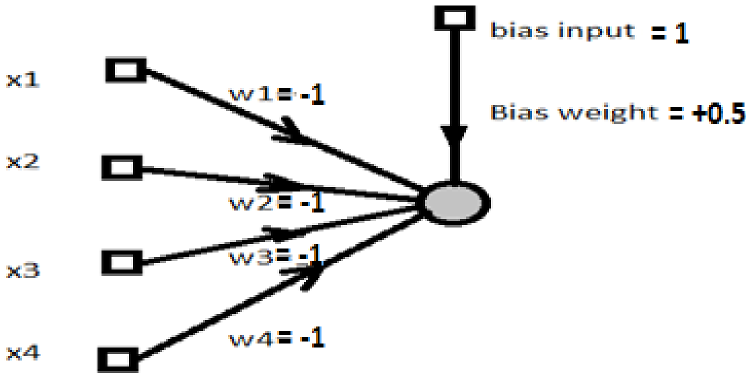
The concept of the logic gate OR is that it may have multiple inputs, but the output will attain a FALSE state if and only if all the "m" inputs in the problem of consideration are FALSE simultaneously. If any input becomes TRUE, the output immediately becomes TRUE. This too is a perfectly linearly separable problem in the m dimensional input space. It resembles the Union of two sets.



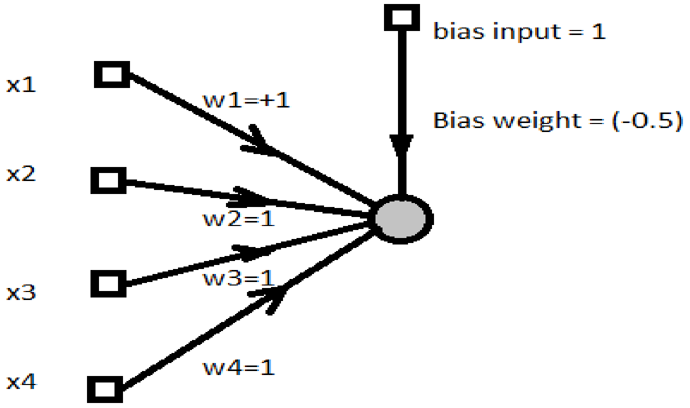
A	B	A+B = OR
0	0	0
0	1	1
1	0	1
1	1	1

Figure 4. The symbol and Truth table of a 2 input OR Gate

The logic of the OR gate is that in the absence of any of the inputs being active, the output should be false. This obviously implies that the bias should be negative if we take a threshold activation function. The output should be false for all FALSE inputs, but when any inputs get active, the output takes a "True" state. That means each of the inputs acting alone should be able to "defeat" the bias and take the output to a "True" state. Hence while the bias can be taken as a negative value, each input should have a positive synaptic weight greater in magnitude than the bias weight. Below in figure 5(a) is a sample circuit for a 4 input OR Gate using SLP neuron.



(a)



(b)

Figure 5. (a) SLP Implementation of OR Gate with 4 inputs taking Bias = -0.5 (b) SLP Implementation of OR Gate with 4 inputs taking Bias = + 0.5

NOR gate is the logical inversion of OR Gate. So just like we did or NAND; we can simply invert all polarities of the OR gate to get the implementation of NOR gate. Alternatively, we can keep polarities same but invert the Activation function. While OR is a basic gate, NOR is a universal gate. Figure 5 (b) shows the 4 input NOR gate as a logical inversion of the 4 input OR Gate in Figure 5 (a).

2.3 NOT Gate

Not gate is also called inverter. It has a single input and when the input is true, the output is false and vice versa.

A	A'
0	1
1	0

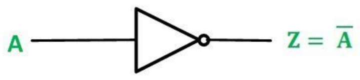


Figure 6. NOT Gate truth table and symbolic representation

As the output is 1 in the absence of any inputs, the bias is negative. Additionally, single input should be able to beat the bias and convert he output to low. Hence input's synaptic weight should be negative and stronger than the bias.

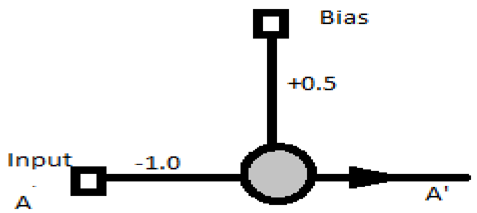


Figure 7. SLP for Inverter gate

2.4 XOR Gate

XOR gate is a special gate, which is not linearly separable with respect to its inputs. It is basically an inequality detector. For a two input XOR, if both inputs are equal (either both zero or both one) he output stays false. But if any one single input is 1, output is zero. As said, the space is not linearly separable. Figure8 shows the gate symbol truth table. We can interpret this to say "XOR IS TRUE IF and ONLY IF (OR IS TRUE) and simultaneously (AND IS FALSE)." Another interpretation is to say "XOR IS TRUE IF (A'B+AB')" The first interpretation will be shown below with HALF ADDER and second one with HALF SUBTRACTER.

A	B	output
0	0	0
0	1	1
1	0	1
1	1	0

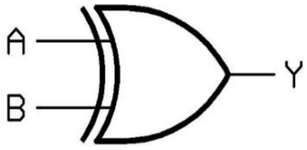


Figure 8. XOR truth table and symbol

We have solved the basic gates above- they are all linearly separable cases. Figure 9 shows the output mapping in 2-dimensional space horizontal and vertical axis indicate the two inputs which may be a zero or a one, and map out a square in the space at origin bound by 4 vertices (0,0), (0,1) (1,0) (1,1)

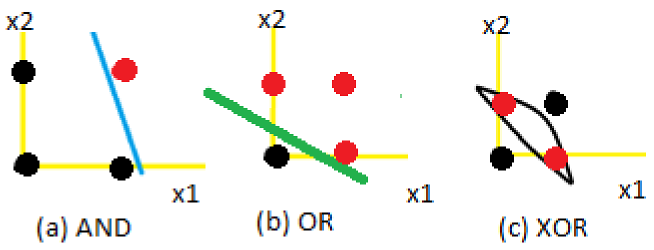


Figure 9. Mapping for (a) AND, (b) OR, (c) XOR. Here Red represents the 1's in the output and black represent the 0's

Figure 10 shows one possible implementation of an XOR Gate. Another logical implementation of XOR gate is the A is not Equal to B. It can be represented as A'B+B'A. Hence the implementation of XOR may be made with this as shown in the Figure 11.

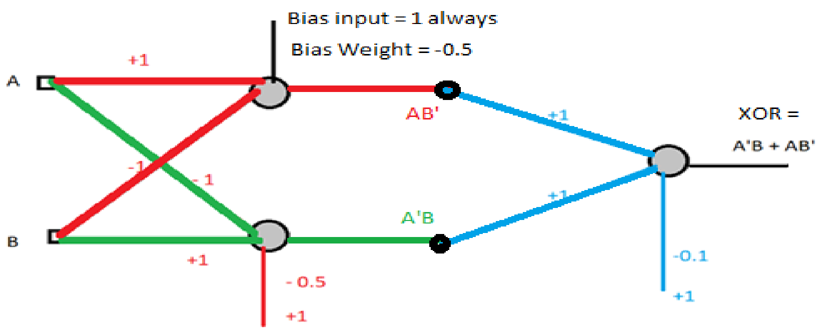


Figure 10. One possible XOR implementation with 2 Layer network with logic "OR but Not AND"

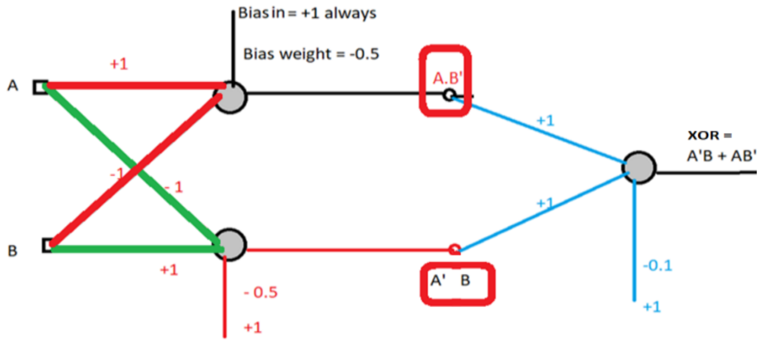


Figure 11. Implementation of XOR with logic "A is not Equal to B"

2.5 XNOR Gate

Converse to the above two implementations of XOR, the XNOR may be represented as "AND or NOR" which is expressed in Boolean algebra as (A+B)'+AB. XNOR can also be implemented from SOP terms as from truth table as A'B'+AB as "A equals B". It is implemented with 2-layer perceptron only he logic is implemented in figure 12 (A+B)'+AB.

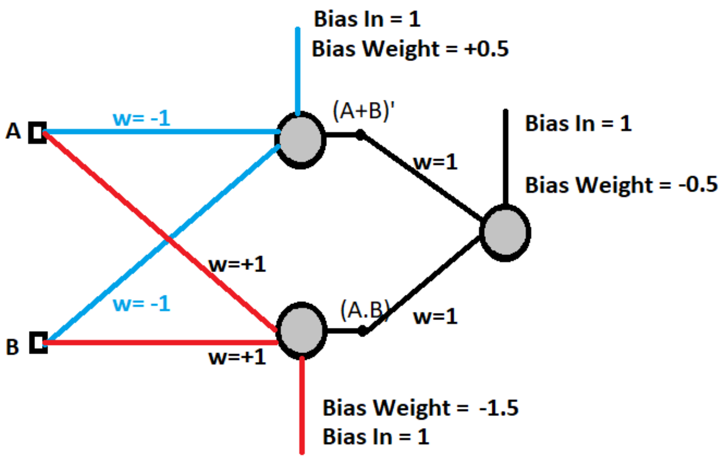


Figure 12. XNOR Implementation

3 Conclusion

All modern computers are made with digital logic gates at their foundation. Digital logic was designed based on Boolean algebra which is the mathematic model for human reasoning. Still, these gates operate only by the On-Off combinations of switching circuits (basically transistors). But Artificial Neural networks work based on biological learning approaches, and as such re highly suited for implementation of logic. This paper tries to use ANN for implementing all Boolean logic gates in place of traditional binary circuits. The performance will be slower than the high-speed switching circuits, but much more suitable to reasoning and self-learning networks.

References

[1] Mano, M. Morris (1979). Digital Logic and Computer Design. Prentice-Hall. pp. 119–123. ISBN 978-0-13-214510-7.

[2] Futurlec.com. 2021. 7483 Technical Data. [online] Available at: <https://www.futurlec.com/74/IC7483.shtml> [Accessed 21 September 2021].

[3] Circuitstoday.com. 2021. [online] Available at: <https://www.circuitstoday.com/flip-flops> [Accessed 21 September 2021].

[4] Education, I., 2021. What are Recurrent Neural Networks? [online] Ibm.com. Available at: <https://www.ibm.com/cloud/learn/recurrent-neural-networks> [Accessed 21 September 2021].

[5] Mehta, S. (2008, March). Fuzzy control system for controlling traffic lights. In Proceedings of the International Multi Conference of Engineers and Computer Scientists (Vol. 1, pp. 19-21).

[6] Medium. 2021. Implementing Logic Gates using Neural Networks (Part 2). [online] Available at: <https://towardsdatascience.com/implementing-logic-gates-using-neural-networks-part -2-b284cc159fce> [Accessed 21 September 2021].

[7] Medium. 2021. Neural Representation of AND, OR, NOT, XOR and XNOR Logic Gates (Perceptron Algorithm). [online] Available at: <https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1> [Accessed 21 September 2021].