Don Bosco Institute of Technology, Mumbai

Department of Computer Engineering

BE Computer 2021 - 22

Experiment No.: 7

Course: **Artificial Intelligence & Soft Computing Lab**

Course Code: **CSL703**

Name: Hayden Cordeiro

Roll No.: 05

Batch: D

Aim: Implement the OR,AND and NOT gates using Mc Culloch Pitt model

Learning Objective: Students have to design Neural Networks using Mc Culloch Pitt Model

Learning Outcome: Student are able to successfully Implement the OR,AND and NOT gates using Mc Culloch Pitt model

CSL703.4 To realize the basic techniques to build intelligent systems

Program Outcome

(PO 3) Design/ development of solutions: Breadth and uniqueness of engineering problems i.e., the extent to which problems are original and to which solutions have previously been identified or codified?

(PO 12) Life Long Learning

Bloom's Taxonomy Level

- Remembering
- Understanding

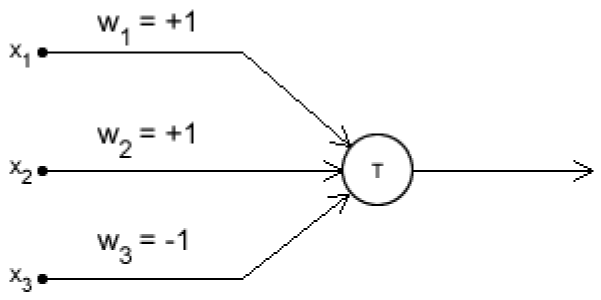
Theory:

The McCulloch-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. And the output was a zero or a one. And each input could be either excitatory or inhibitory.

Now the whole point was to sum the inputs. If an input is one, and is excitatory in nature, it added one. If it was one, and was inhibitory, it subtracted one from the sum. This is done for all inputs, and a final sum is calculated.

Now, if this final sum is less than some value (which you decide, say T), then the output is zero. Otherwise, the output is a one.

Here is a graphical representation of the McCulloch-Pitts model



In the figure, I represented things with named variables. The variables w_1, w_2 and w_3 indicate which input is excitatory, and which one is inhibitory. These are called "weights". So, in this model, if a weight is 1, it is an excitatory input. If it is -1, it is an inhibitory input.

x_1, x_2, and x_3 represent the inputs. There could be more (or less) inputs if required. And accordingly, there would be more 'w's to indicate if that particular input is excitatory or inhibitory.

Now, if you think about it, you can calculate the sum using the 'x's and 'w's... something like this:

$\text{sum} = x_1w_1 + x_2w_2 + x_3w_3 + \ldots$

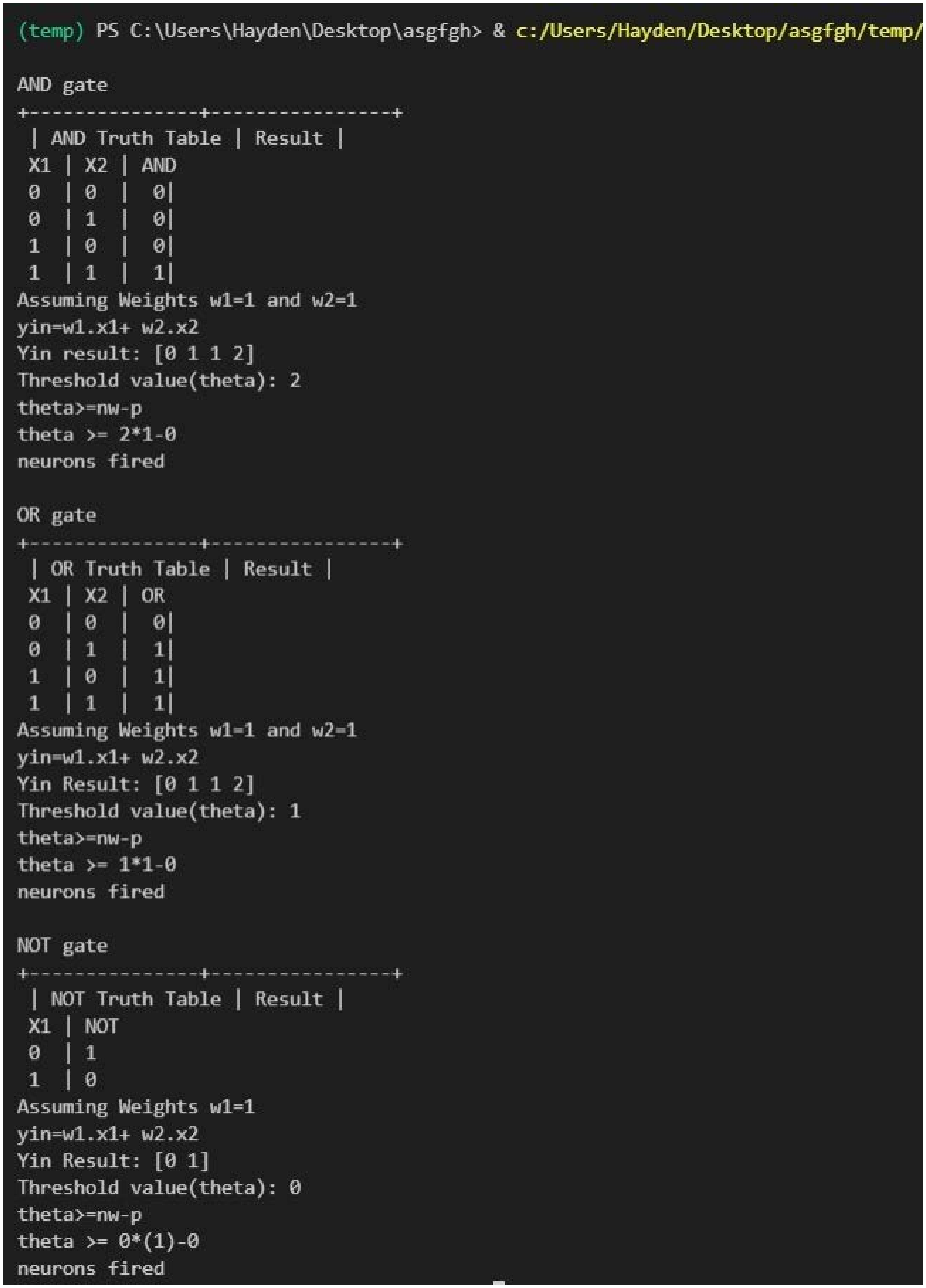
This is what is called a 'weighted sum'.

Now that the sum has been calculated, we check if sum < T or not. If it is, then the output is made zero. Otherwise, it is made a one.

Code

```
import numpy as np

def linear_threshold_gate(dot: int, T: float) -> int:
    '''Returns the binary threshold output'''
    if dot >= T:
        return 1
    else:
        return 0

# matrix of inputs
input_table = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])

#AND gate
print('\nAND gate')
print("+---------------+----------------+")
print(" | AND Truth Table | Result |")
print(" X1 | X2 | AND")
print(" 0  | 0  |  0| ")
print(" 0  | 1  |  0| ")
print(" 1  | 0  |  0| ")
print(" 1  | 1  |  1| ")
weights = np.array([1,1])
print("Assuming Weights w1=1 and w2=1")
dot_products = input_table @ weights
print("yin=w1.x1+ w2.x2")
print(f'Yin result: {dot_products}')
T = 2
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
res=(2*1-0)
print("Threshold value(theta):",T)
print("theta>=nw-p")
print("theta >= 2*1-0")
if(T >= res ):
    print("neurons fired")
else:
    print("neurons not fired")

#OR gate
print('\nOR gate')
print("+---------------+----------------+")
print(" | OR Truth Table | Result |")
print(" X1 | X2 | OR")
print(" 0  | 0  |  0| ")
print(" 0  | 1  |  1| ")
print(" 1  | 0  |  1| ")
print(" 1  | 1  |  1| ")
weights = np.array([1,1])
print("Assuming Weights w1=1 and w2=1")
print("yin=w1.x1+ w2.x2")
dot_products = input_table @ weights
print(f'Yin Result: {dot_products}')
T1 = 1
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T1)
res1=(1*1-0)
print("Threshold value(theta):",T1)
print("theta>=nw-p")
print("theta >= 1*1-0")
if(T1 >= res1 ):
    print("neurons fired")
else:
    print("neurons not fired")
input_table2 = np.array([
    [0],
    [1]
])
#NOT gate
input_table = np.array([
    [0],
    [1]
])

print('\nNOT gate')
print("+---------------+---------------+")
print("  | NOT Truth Table | Result |")
print(" X1 | NOT")
print(" 0  | 1")
print(" 1  | 0")
weights = np.array([1])
print("Assuming Weights w1=1")
dot_products = input_table @ weights
print("yin=w1.x1+ w2.x2")
print(f'Yin Result: {dot_products}')
T2 = 0
for i in range(0,2):
    activation = linear_threshold_gate(dot_products[i], T2)
res2=(0*1-0)
print("Threshold value(theta):",T2)
print("theta>=nw-p")
print("theta >= 0*(1)-0")
if(T2 >= res2 ):
    print("neurons fired")
else:
    print("neurons not fired")
```

Output



```
(temp) PS C:\Users\Hayden\Desktop\asgfgh> & c:/Users/Hayden/Desktop/asgfgh/temp/

AND gate
+---------------+---------------+
 | AND Truth Table | Result |
 X1 | X2 | AND
 0  | 0  |  0|
 0  | 1  |  0|
 1  | 0  |  0|
 1  | 1  |  1|
Assuming Weights w1=1 and w2=1
yin=w1.x1+ w2.x2
Yin result: [0 1 1 2]
Threshold value(theta): 2
theta>=nw-p
theta >= 2*1-0
neurons fired

OR gate
+---------------+---------------+
 | OR Truth Table | Result |
 X1 | X2 | OR
 0  | 0  |  0|
 0  | 1  |  1|
 1  | 0  |  1|
 1  | 1  |  1|
Assuming Weights w1=1 and w2=1
yin=w1.x1+ w2.x2
Yin Result: [0 1 1 2]
Threshold value(theta): 1
theta>=nw-p
theta >= 1*1-0
neurons fired

NOT gate
+---------------+---------------+
 | NOT Truth Table | Result |
 X1 | NOT
 0  | 1
 1  | 0
Assuming Weights w1=1
yin=w1.x1+ w2.x2
Yin Result: [0 1]
Threshold value(theta): 0
theta>=nw-p
theta >= 0*(1)-0
neurons fired
```

Conclusion: The logic gates are successfully implemented using Neural network package in python