

| **Title:** Implementation of ‘McCulloch Pitts Net for NAND and AND NOT function |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** To implement MP Neuron Model for the NAND and AND NOT logical function

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

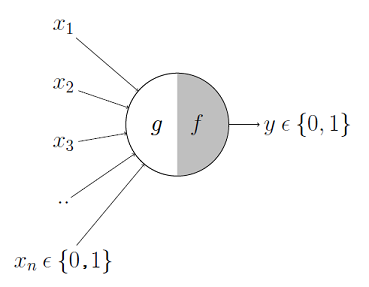
CO1 : Identify and describe soft computing techniques and their roles **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Pre Lab/ Prior Concepts:**

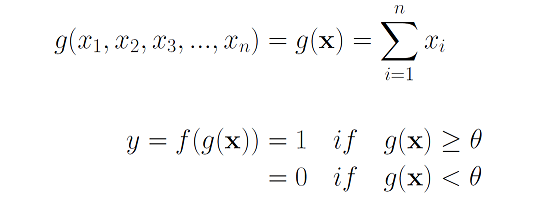
**McCulloch-Pitts Neuron**

The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.



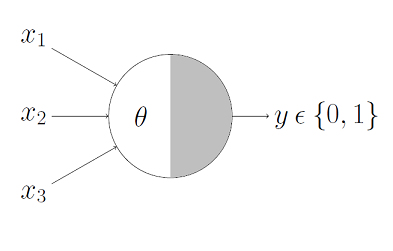
It may be divided into 2 parts. The first part, ***g***takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, ***f*** makes a decision.

These inputs can either be *excitatory* or *inhibitory*. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs i.e., if the inhibitory input is ON than the neuron will never fire. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:



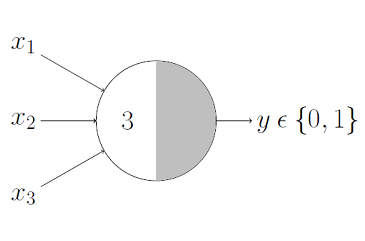
We can see that ***g*(x)** is just doing a sum of the inputs — a simple aggregation. And ***theta*** here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the ***theta***is 2 here. This is called the Thresholding Logic.

**Boolean Functions Using M-P Neuron**



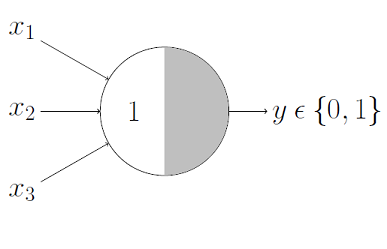
This representation just denotes that, for the boolean inputs ***x\_1***, ***x\_2*** and ***x\_3*** if the ***g*(x)** i.e., **sum** **≥** **theta**, the neuron will fire otherwise, it won’t.

**AND Function**



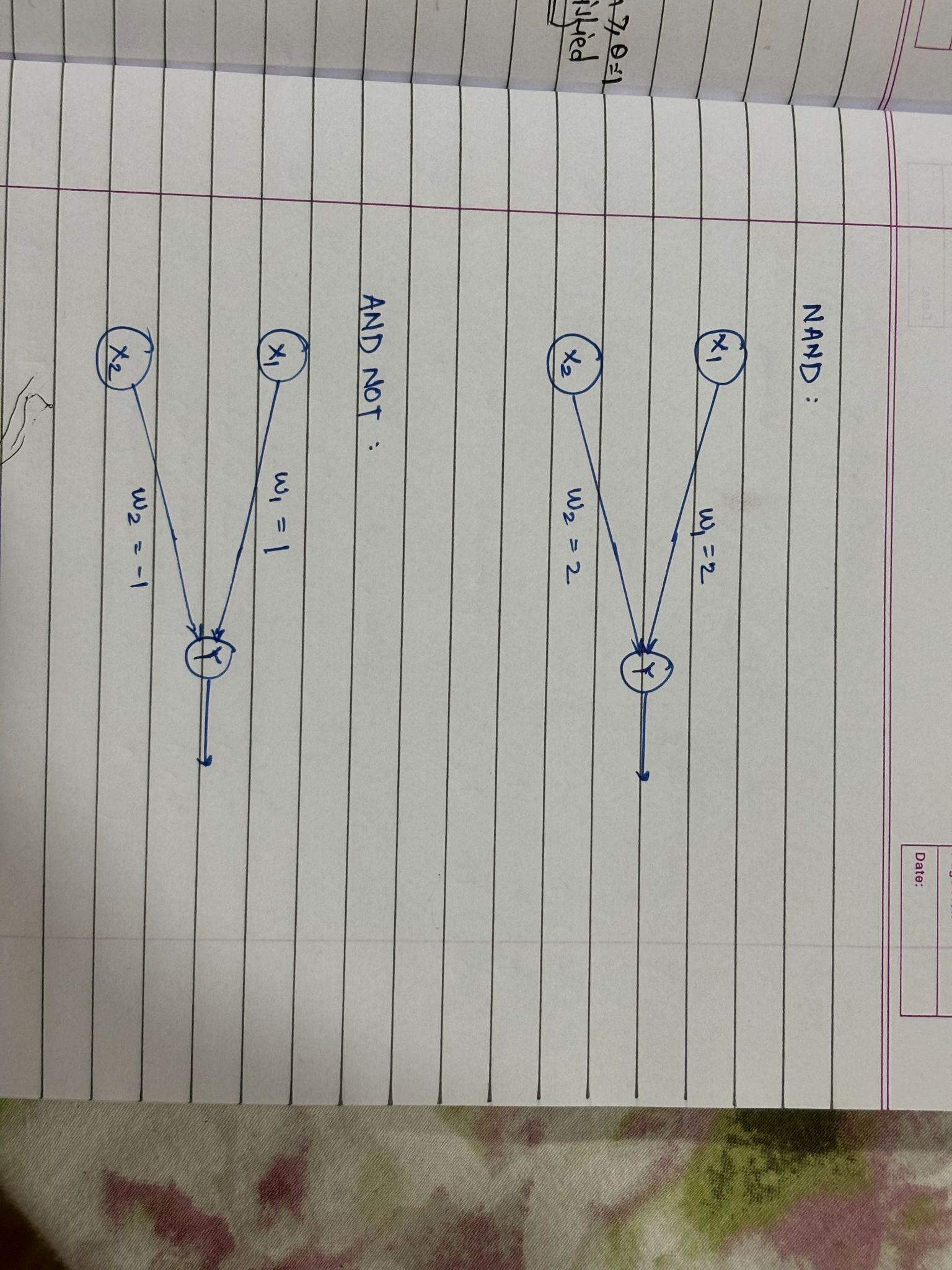
An AND function neuron would only fire when ALL the inputs are ON i.e., ***g*(x)** ≥ 3 here.

**OR Function**

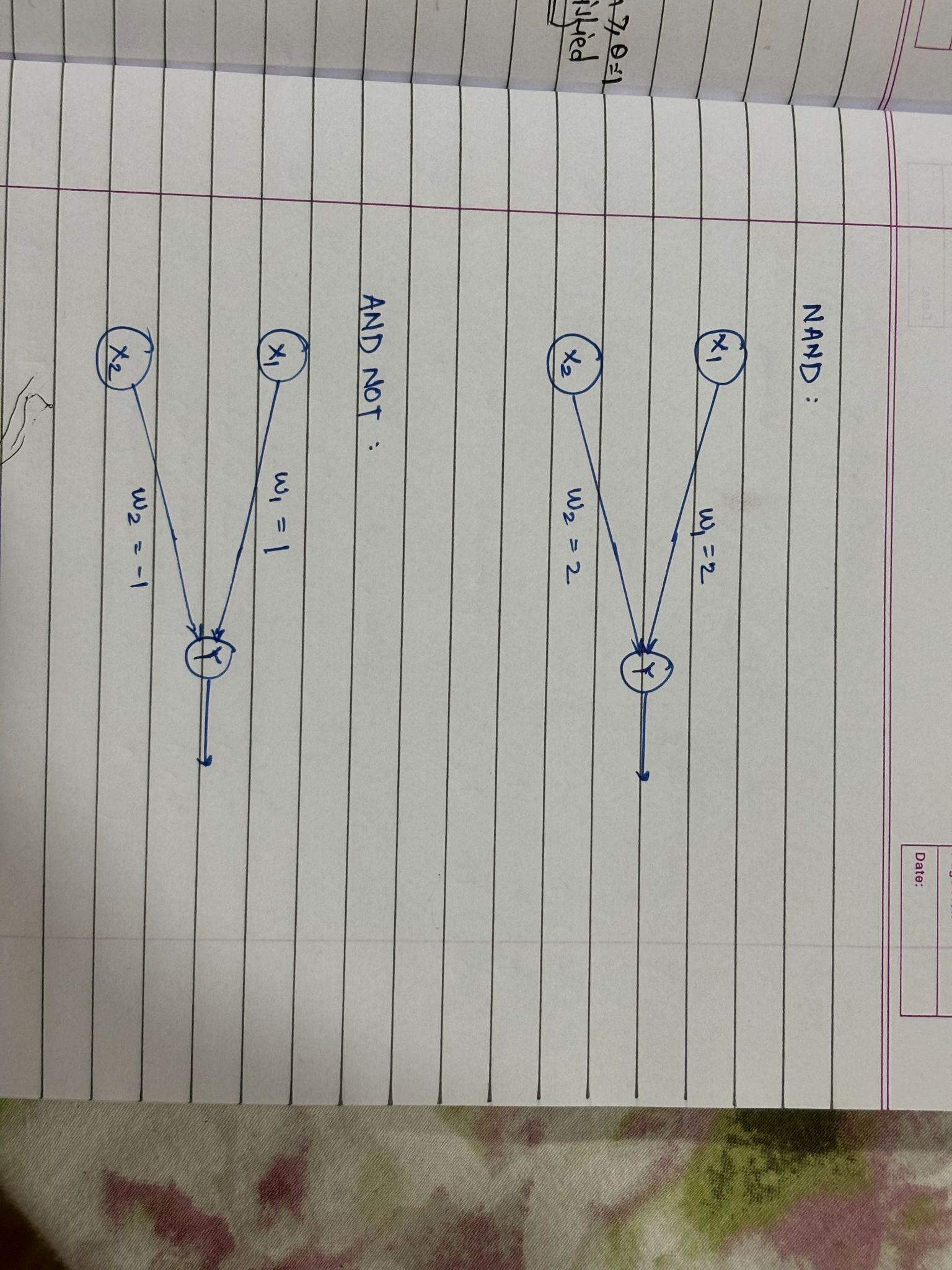


I believe this is self explanatory as we know that an OR function neuron would fire if ANY of the inputs is ON i.e., ***g*(x)** ≥ 1 here.

**AND NOT function**



**NAND function (write here)**

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Implementation Details:**

**Write the code for implementation of MP neuron model for NAND and AND NOT function**

**Code:**

import numpy as np

def mc\_culloch\_pitts(weights, inputs, threshold):

weighted\_sum = np.dot(weights, inputs)

return 1 if weighted\_sum >= threshold else 0

def not\_function(x):

weights = [-1]

threshold = 0

return mc\_culloch\_pitts(weights, [x], threshold)

def nand\_function(x1, x2):

and\_weights = [1, 1]

and\_threshold = 2

and\_output = mc\_culloch\_pitts(and\_weights, [x1, x2], and\_threshold)

return not\_function(and\_output)

def and\_not\_function(x1, x2):

and\_weights = [1, 1]

and\_threshold = 2

not\_x1 = not\_function(x1)

return mc\_culloch\_pitts(and\_weights, [not\_x1, x2], and\_threshold)

print("NAND function outputs:")

for x1 in [0, 1]:

for x2 in [0, 1]:

print(f"nand({x1}, {x2}) = {nand\_function(x1, x2)}")

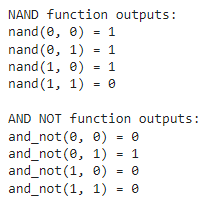
print("\nAND NOT function outputs:")

for x1 in [0, 1]:

for x2 in [0, 1]:

print(f"and\_not({x1}, {x2}) = {and\_not\_function(x1, x2)}")

**Output:**



**Conclusion:** Thus, we have successfully implemented MP neuron model for NAND and AND NOT logical function

**Post Lab Descriptive Questions :**

1. Discuss the limitations of MP neuron model

Binary Output: The MP neuron model produces a binary output (0 or 1), which limits its ability to handle more complex, real-valued data. Real-world data often involves continuous values rather than discrete binary classifications.

Linearity: The MP neuron model only implements linear decision boundaries. It uses a linear combination of inputs and a step function to produce an output, which restricts it to solving linearly separable problems.

Lack of Learning: The basic MP neuron model does not include any learning mechanism. It does not adapt its weights based on input-output pairs. This means it cannot improve or adjust its behavior based on experience or feedback.

Limited Computational Power: The MP neuron model is not capable of implementing complex functions or representing non-linear relationships between inputs and outputs. This limits its practical applicability in solving many real-world problems.

No Memory: MP neurons do not have memory or feedback mechanisms. They process each input independently without considering past information, which restricts their utility in tasks requiring temporal dynamics or sequence processing.

1. Explain with an example the concept of linear separability and Justify NOT XOR function is not linearly separable.

Linear Separability is a way to understand if you can separate different groups of data with a straight line (in 2D) or a flat plane (in 3D).

Example:

Imagine you have two groups of dots on a piece of paper:

Group A: Dots at (1, 2) and (2, 3)

Group B: Dots at (4, 5) and (5, 6)

If you can draw a straight line that puts all dots of Group A on one side and all dots of Group B on the other side, then the data is linearly separable. In this case, you can draw such a line, so the data is separable by a straight line.

XOR Function and Why It’s Not Linearly Separable

The XOR function is a simple mathematical rule with four possible input combinations:

(0, 0) results in 0

(0, 1) results in 1

(1, 0) results in 1

(1, 1) results in 0

If you plot these points on a graph, you get:

0 points at (0, 0) and (1, 1)

1 points at (0, 1) and (1, 0)

If you try to draw a straight line to separate these two types of points, you will find it's impossible. The points of the two types are arranged in such a way that you need a curved boundary, not a straight line, to separate them correctly.

You can't draw a single straight line to separate the “0” points from the “1” points. They are mixed up in a way that requires a more complex boundary than just a straight line. This is why the XOR function is not linearly separable.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**