

| **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:**

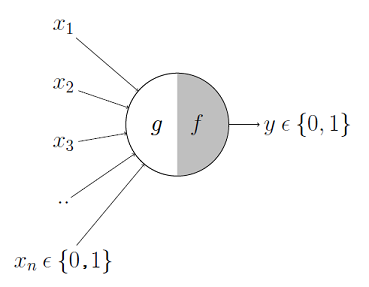
CO1 : Identify and describe soft computing techniques and their roles **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

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

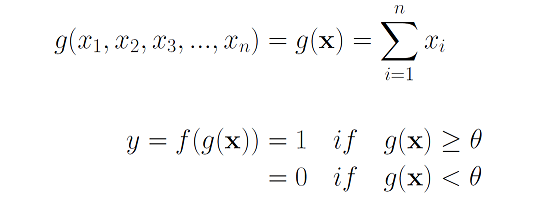
**McCulloch-Pitts Neuron**

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



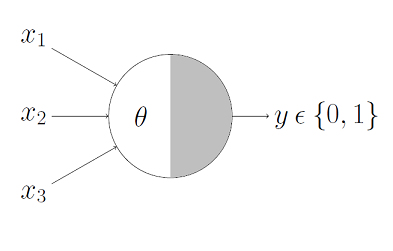
It may be divided into 2 parts. The first part, ***g***takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, ***f*** makes a decision.

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:



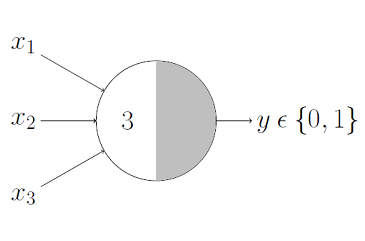
We can see that ***g*(x)** is just doing a sum of the inputs — a simple aggregation. And ***theta*** here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the ***theta***is 2 here. This is called the Thresholding Logic.

**Boolean Functions Using M-P Neuron**



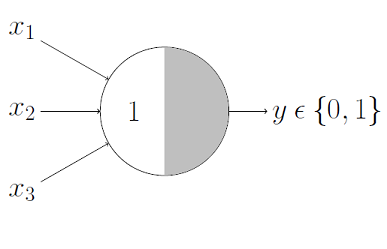
This representation just denotes that, for the boolean inputs ***x\_1***, ***x\_2*** and ***x\_3*** if the ***g*(x)** i.e., **sum** **≥** **theta**, the neuron will fire otherwise, it won’t.

**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 (write here)**

**NAND function (write here)**

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

**Implementation Details:**

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

**Code:**

**NAND Function:**

def mcculloch\_pitts\_nand\_3input(input1, input2, input3):

# Weights for the inputs

w1 = -1

w2 = -1

w3 = -1

# Threshold

theta = -2.5

# Calculate the weighted sum

weighted\_sum = w1 \* input1 + w2 \* input2 + w3 \* input3

# Apply the threshold function

if weighted\_sum > theta:

return 1

else:

return 0

# Function to get binary input from user

def get\_binary\_input(prompt):

while True:

try:

value = int(input(prompt))

if value in [0, 1]:

return value

else:

print("Please enter 0 or 1.")

except ValueError:

print("Invalid input. Please enter 0 or 1.")

# Get user inputs for the three inputs

input1 = get\_binary\_input("Enter the first input (0 or 1): ")

input2 = get\_binary\_input("Enter the second input (0 or 1): ")

input3 = get\_binary\_input("Enter the third input (0 or 1): ")

# Get the output of the three-input NAND gate

output = mcculloch\_pitts\_nand\_3input(input1, input2, input3)

# Display the result

print(f"Three-Input NAND Gate Output for inputs ({input1}, {input2}, {input3}): {output}")

**AND NOT Function:**

def mcculloch\_pitts\_and\_not\_3input(input1, input2, input3):

# Weights for the inputs

w1 = 1 # AND condition (must be true)

w2 = -1 # NOT condition (must be false)

w3 = -1 # NOT condition (must be false)

# Threshold

theta = 0.5

# Calculate the weighted sum

weighted\_sum = w1 \* input1 + w2 \* input2 + w3 \* input3

# Apply the threshold function

if weighted\_sum > theta:

return 1

else:

return 0

# Function to get binary input from user

def get\_binary\_input(prompt):

while True:

try:

value = int(input(prompt))

if value in [0, 1]:

return value

else:

print("Please enter 0 or 1.")

except ValueError:

print("Invalid input. Please enter 0 or 1.")

# Get user inputs for the three inputs

input1 = get\_binary\_input("Enter the first input (0 or 1): ")

input2 = get\_binary\_input("Enter the second input (0 or 1): ")

input3 = get\_binary\_input("Enter the third input (0 or 1): ")

# Get the output of the three-input AND NOT gate

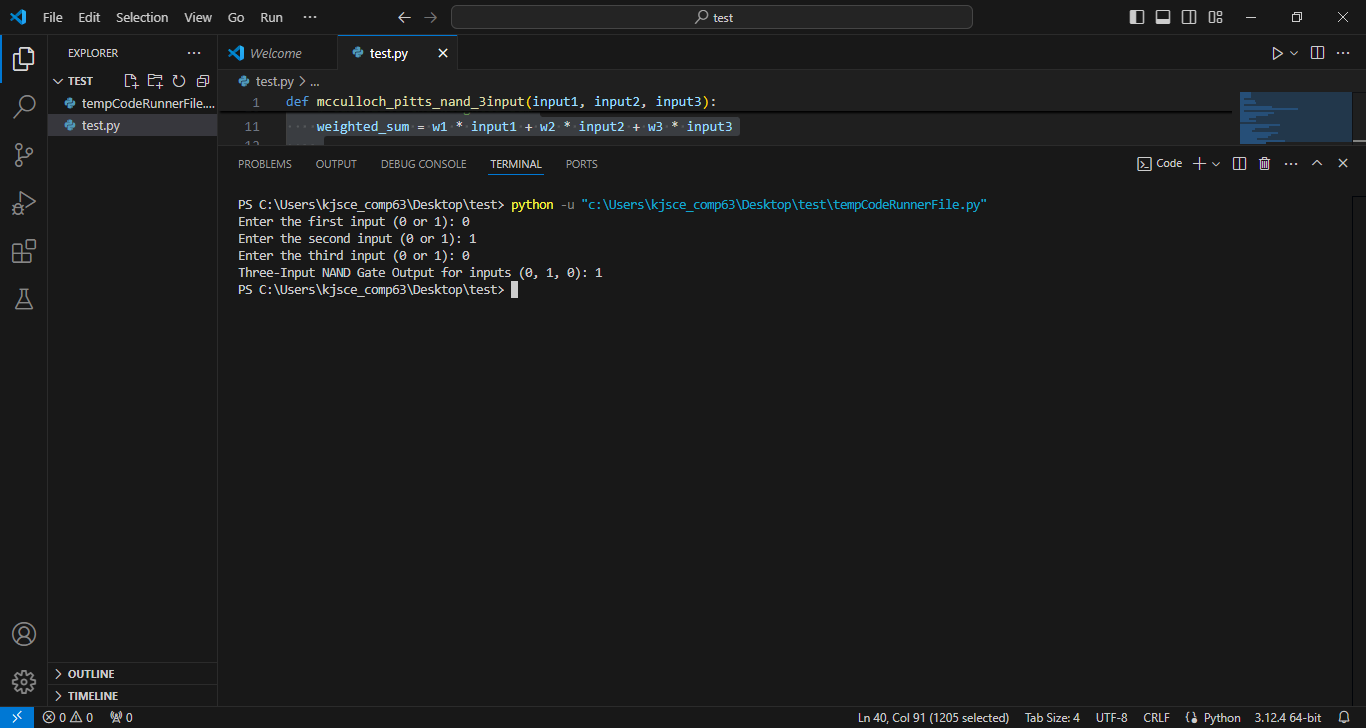
output = mcculloch\_pitts\_and\_not\_3input(input1, input2, input3)

# Display the result

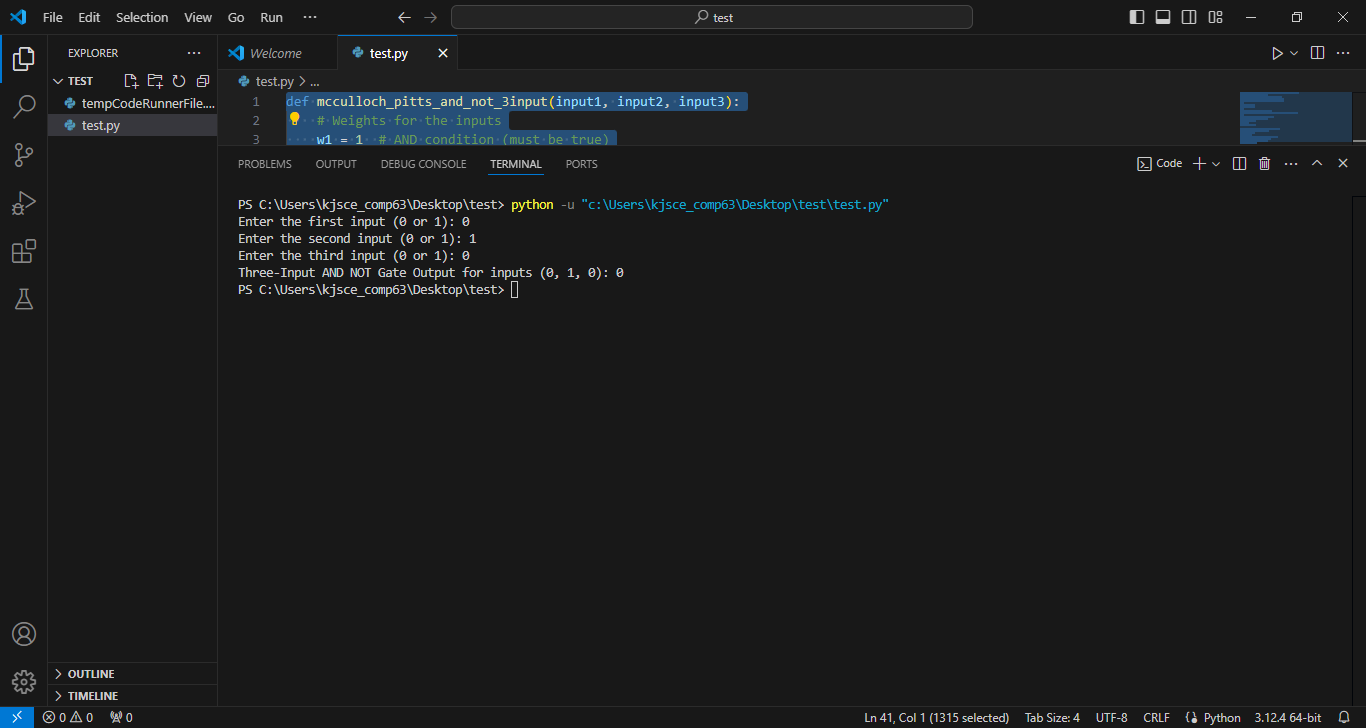
print(f"Three-Input AND NOT Gate Output for inputs ({input1}, {input2}, {input3}): {output}")

**Output:**

**AND Function:**

****

**AND NOT Function:**

****

**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

**Ans:**

**The McCulloch-Pitts (MP) neuron model was one of the first computational models for neurons, introduced in 1943. Although it laid the groundwork for modern neural networks, it has several limitations:**

**Binary Inputs and Outputs: The MP neuron model only accepts binary inputs (0 or 1) and produces binary outputs. This restricts its ability to model more complex behaviors where continuous values are needed.**

**Linear Classification: The MP neuron can only solve problems that are linearly separable. It cannot solve more complex problems, like XOR, which are not linearly separable.**

**Fixed Threshold: The threshold in the MP neuron is fixed and does not adapt or change. This limits the model’s flexibility in handling varying types of inputs.**

**Lack of Learning: The MP model does not include a learning algorithm. The weights and thresholds are predetermined and do not change, making it impossible for the neuron to learn from data.**

**No Temporal Processing: The MP neuron processes inputs statically, meaning it doesn’t handle time-dependent data. Real neurons, however, process information over time, which is critical for many cognitive tasks.**

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

**Ans:**

**Linear Separability:**

**Linear separability is a concept in geometry and machine learning. A set of data points is said to be linearly separable if there exists a straight line (or a hyperplane in higher dimensions) that can separate the data points into distinct classes without any misclassification.**

**If a problem is linearly separable, a single linear classifier (like a perceptron) can be used to correctly classify the inputs.**

**Example:**

**Consider the AND function with two binary inputs. The possible input-output pairs are:**

**(0, 0) -> 0**

**(0, 1) -> 0**

**(1, 0) -> 0**

**(1, 1) -> 1**

**These points can be plotted on a 2D plane, where the inputs are the x and y coordinates, and the output is the class. The AND function is linearly separable because you can draw a straight line that separates the (1, 1) point from the others.**

**NOT XOR Function and Non-Linearity:**

**The XOR function, however, is not linearly separable. The input-output pairs for XOR are:**

**(0, 0) -> 0**

**(0, 1) -> 1**

**(1, 0) -> 1**

**(1, 1) -> 0**

**If you plot these points, you'll see that there is no single straight line that can separate the points where the output is 1 from the points where the output is 0. This is because the XOR function is inherently nonlinear.**

**Justification:**

**For XOR, the positive outputs (1) are diagonally across from each other in the input space, making it impossible to separate them from the negative outputs (0) using a single line.**

**Since the XOR problem is not linearly separable, a single MP neuron (or perceptron) cannot solve it. This is why more complex models like multi-layer perceptrons (with hidden layers) are needed to solve XOR, as they can create non-linear decision boundaries.**

**Date: 20/08/2024 Signature of faculty in-charge**