Experiment # 11: Implement 3 input AND, OR, NAND and NOR Boolean functions using the McCulloch-Pitts Model using Python

1.what is a neural network? Explain the analogy between artificial neurons and biological neurons.

A neural network is a machine learning model inspired by the human brain. Artificial neurons receive inputs, process them with weights, and produce outputs—similar to biological neurons using dendrites, axons, and synapses.

---

2. What is the McCulloch–Pitts model? How does it simulate the behavior of a neuron using binary inputs and a threshold function?

The McCulloch–Pitts model simulates a neuron using binary inputs and a threshold. If the sum of inputs meets or exceeds the threshold, the neuron fires (outputs 1); otherwise, it outputs 0.

---

3. Provide examples of Boolean functions (e.g., AND, OR) and explain how they can be implemented using the McCulloch–Pitts model.

- **AND Gate**: Inputs (1,1), threshold = 2 → Output = 1
- **OR Gate**: Any input = 1, threshold = 1 → Output = 1
  Implemented by summing inputs and comparing with a threshold.

---

4. Discuss McCulloch–Pitts model and Perceptron model with suitable diagram.

- **McCulloch–Pitts**: Binary inputs/outputs, fixed weights, threshold logic
- **Perceptron**: Real-valued inputs/weights, learns through training
  Diagram shows input → weights → sum → activation → output.

---

5. Describe the steps involved in implementing a basic neural network model in Python using libraries like NumPy or TensorFlow.

1. Import necessary libraries (NumPy/TensorFlow)
2. Define input and output data
3. Initialize weights and bias

4. Apply activation function (e.g., sigmoid)
5. Use training loop and update weights using loss function and backpropagation

Implement 3 input AND, OR, NOR and NAND Boolean functions using McCulloch−Pitts Model using python programming language.

Program:

```
inputs = [
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 1],
    [1, 0, 0],
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
]
def mcp_model(input_set, threshold):
    return [1 if sum(x) >= threshold else 0 for x in input_set]
and_threshold = 3
or_threshold = 1
nand_threshold = 3  # Inverted AND
nor_threshold = 1   # Inverted OR
and_output = mcp_model(inputs, and_threshold)
or_output = mcp_model(inputs, or_threshold)
nand_output = [1 - y for y in and_output]
nor_output = [1 - y for y in or_output]
```

```
print("a1 a2 a3 | AND OR NAND NOR")

for i, input_set in enumerate(inputs):

   print(f"{input_set[0]} {input_set[1]} {input_set[2]} | {and_output[i]}  {or_output[i]}
{nand_output[i]}   {nor_output[i]}")
```

Data and Results:

a_1	a_2	a_3	Ysum	AND	OR	NAND	NOR
0	0	0	0	0	0	1	1
0	0	1	1	0	1	1	0
0	1	0	1	0	1	1	0
0	1	1	2	0	1	1	0
1	0	0	1	0	1	1	0
1	0	1	2	0	1	1	0
1	1	0	2	0	1	1	0
1	1	1	3	1	1	0	0

Analysis and Inferences:

- The **McCulloch–Pitts Model** correctly simulates Boolean logic using threshold values.
- For **AND gate**, the neuron fires only when all inputs are 1 (i.e., sum = 3).
- For **OR gate**, the neuron fires if any one input is 1 (i.e., sum ≥ 1).
- **NAND** and **NOR** are simply the inverses of AND and OR outputs, respectively.
- This model demonstrates how basic logic gates can be built using simple threshold logic, showing the foundation for more complex neural networks.

1. What is the McCulloch–Pitts model, and how is it used to implement Boolean functions?

The McCulloch–Pitts model is a simplified mathematical model of a neuron that uses binary inputs and a threshold function to decide the output. It is used to implement Boolean functions by setting appropriate weights and thresholds.

---

2. How do you implement the AND and OR Boolean functions using the McCulloch–Pitts model?

- **AND**: Set all weights to 1 and threshold = number of inputs (e.g., 3 for 3-input AND).
- **OR**: Set weights to 1 and threshold = 1 (fires if any one input is 1).

---

3. What changes are needed to extend the McCulloch–Pitts model to implement NOR and NAND Boolean functions?

To implement **NAND** and **NOR**, first implement AND or OR using the McCulloch–Pitts model, then **invert the output** (i.e., output = 1 − original output).

---

4. How do you validate the correctness of your McCulloch–Pitts model implementation for Boolean functions?

By comparing the output of the model for all input combinations with the **truth table** of the target Boolean function.

---

5. What are the limitations of the McCulloch–Pitts model in representing complex Boolean functions?

- It uses **binary inputs and outputs only**
- Cannot handle **non-linearly separable functions** like XOR
- No learning mechanism or weight updates
- Only supports **fixed thresholds**

Implement Perceptron model to solve AND and OR gates with bias and weights.
Program:

```
import numpy as np
def step(x):
    return 1 if x >= 0 else 0
def perceptron(inputs, weights, bias):
    output = []
    for x in inputs:
        y = np.dot(x, weights) + bias
        output.append(step(y))
    return output
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
weights_and = np.array([1, 1])
bias_and = -1.5  # sum must be >= 1.5 to output 1 → only when both inputs are 1
and_output = perceptron(X, weights_and, bias_and)
weights_or = np.array([1, 1])
bias_or = -0.5  # sum must be >= 0.5 to output 1 → when any one input is 1
or_output = perceptron(X, weights_or, bias_or)
print("X1 X2 | AND OR")
for i in range(len(X)):
    print(f"{X[i][0]}  {X[i][1]}  |  {and_output[i]}   {or_output[i]}")
```

Data and Results:

X_1	X_2	Weighted Sum (AND)	Output (AND)	Weighted Sum (OR)	Output (OR)
0	0	-1.5	0	-0.5	0
0	1	-0.5	0	0.5	1
1	0	-0.5	0	0.5	1
1	1	0.5	1	1.5	1

Analysis and Inferences:

- The **Perceptron model** correctly learns **AND** and **OR** gates using proper weights and bias.
- For **AND**, the output is 1 only when both inputs are 1.
- For **OR**, the output is 1 when any input is 1.
- This shows that Perceptrons can model **linearly separable** functions like AND and OR.
- However, it **cannot model XOR**, which is not linearly separable.