Experiment 5

Title: Implement McCulloch Pitts neural network using Tensorflow.

Aim: To Implement McCulloch Pitts neural network using Tensorflow.

Theory:

McCulloch Pitts neural network

The McCulloch-Pitts neural network is a simplified model of a biological neuron, introduced by Warren McCulloch and Walter Pitts in 1943. This foundational model in neural network theory operates using binary logic (0 or 1) and consists of four main components:

- Inputs: Binary signals (0 or 1) received from other neurons or external stimuli.
- Weights: Assigned to each input to represent the strength of the connection, which can be positive or negative.
- Threshold: A value that determines if the neuron fires an output.
- Activation Function: It sums the weighted inputs and compares the sum to the threshold. If the sum is greater than or equal to the threshold, the neuron outputs 1; otherwise, it outputs 0.

This model can perform simple binary computations and is a basic building block for more complex networks like percept

In [1]:
```
class McCullochPittsNeuron:
    def __init__(self, weights, threshold):
        self.weights = weights
        self.threshold = threshold

    def activate(self, inputs):
        if len(inputs) != len(self.weights):
            raise ValueError("Number of inputs should match the number of weights")
        result = sum([inputs[i] * self.weights[i] for i in range(len(inputs))])
        return 1 if result >= self.threshold else 0

def main():
    # Define the weights and threshold for the AND gate
    and_gate_weights = [1, 1]
    and_gate_threshold = 2

    # Create a McCulloch-Pitts neuron for the AND gate
    and_gate_neuron = McCullochPittsNeuron(and_gate_weights, and_gate_threshold)

    # Test the AND gate
    inputs = [
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1],
    ]

    for input_pair in inputs:
        output = and_gate_neuron.activate(input_pair)
        print(f"{input_pair} -> {output}")

if __name__ == "__main__":
    main()
```

```
[0, 0] -> 0
[0, 1] -> 0
[1, 0] -> 0
[1, 1] -> 1
```

Single Layer Neural Network:

A single-layer neural network, or single-layer perceptron, is a basic neural network architecture that consists of a single layer of interconnected neurons. It is primarily used for simple linear classification tasks.

- Inputs: Similar to the McCulloch-Pitts model, the perceptron receives inputs (features) from the external environment, each associated with a weight.
- Weights: The inputs are multiplied by their corresponding weights, and the weighted inputs are summed. Mathematically:
  Weighted Sum = Σ(w_i * x_i)
- Bias: A bias term is added to the weighted sum. This helps adjust the activation threshold, allowing the network to better control when a neuron fires.
- Activation Function: The weighted sum (with bias) is passed through an activation function, such as a step function, sigmoid, or ReLU. The activation function decides the neuron's output based on the computed value.
- Output = Activation_Function(Weighted Sum + Bias)

In [3]:

```python
import numpy as np

class SingleLayerPerceptron:
    def __init__(self, input_size):
        self.weights = np.random.rand(input_size)
        self.bias = np.random.rand()
        self.learning_rate = 0.1

    def predict(self, inputs):
        net_input = np.dot(inputs, self.weights) + self.bias
        return 1 if net_input >= 0 else 0

    def train(self, training_data, epochs):
        for epoch in range(epochs):
            errors = 0
            for inputs, label in training_data:
                prediction = self.predict(inputs)
                error = label - prediction
                self.weights += self.learning_rate * error * inputs
                self.bias += self.learning_rate * error
                errors += abs(error)
            if errors == 0:
                print(f"Converged after {epoch + 1} epochs.")
                break

def main():
    # Sample training data for OR gate
    training_data = [
        (np.array([0, 0]), 0),
        (np.array([0, 1]), 1),
        (np.array([1, 0]), 1),
        (np.array([1, 1]), 1)
    ]

    input_size = len(training_data[0][0])
    perceptron = SingleLayerPerceptron(input_size)

    epochs = 100
    perceptron.train(training_data, epochs)

    # Test the trained perceptron
    test_data = [
        np.array([0, 0]),
        np.array([0, 1]),
        np.array([1, 0]),
        np.array([1, 1]),
    ]
```

```
    print("Testing the perceptron:")
    for inputs in test_data:
        prediction = perceptron.predict(inputs)
        print(f"{inputs} -> {prediction}")

if __name__ == "__main__":
    main()
```

```
Converged after 3 epochs.
Testing the perceptron:
[0 0] -> 0
[0 1] -> 1
[1 0] -> 1
[1 1] -> 1
```

Output: The final output is determined by the activation function applied to the weighted sum of inputs.

This model is suitable for linear classification but struggles with more complex tasks requiring non-linear separability

Conclusion: Successful Implement McCulloch Pitts neural network using Tensorflow.