Experiment No. 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 neuron, also known as the McCulloch-Pitts model, is a simplified mathematical model of a biological neuron. It was introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neuron is a foundational concept in neural network theory and serves as the basis for understanding more complex artificial neural networks.

The McCulloch-Pitts neuron operates using binary logic (0 or 1) and consists of the following components:

1. Inputs: Each neuron receives binary inputs (0 or 1) from various sources, which represent the incoming
   signals from other neurons or external stimuli.

2. Weights: Each input is assigned a weight that represents the strength of the connection between the input
   and the neuron. The weights can be positive or negative.

3. Threshold: The neuron has a threshold value that determines the level of activation required for it to fire
   an output signal.

4. Activation Function: The McCulloch-Pitts neuron applies a simple activation function. It sums the products of inputs and their corresponding weights and compares this sum to the threshold. If the sum is greater than or equal to the threshold, the neuron outputs 1; otherwise, it outputs 0.

   Mathematically, the output of the McCulloch-Pitts neuron can be represented as:

   Output = 1 if $\Sigma(w_i * x_i) \geq \theta$ (threshold)
   Output = 0 if $\Sigma(w_i * x_i) < \theta$

This basic neuron model can be used to create logical circuits and perform simple binary computations. It serves as a building block for more complex neural network architectures, such as perceptrons and multi layer perceptrons, which have enhanced capabilities for pattern recognition and classification.

Single Layer Neural Network:

A single-layer neural network, often referred to as a single-layer perceptron, is a type of artificial neural network architecture that consists of only one layer of interconnected neurons. It is a basic and elementary form of a neural network, and it can be used for simple linear classification tasks.

Here's how a single-layer perceptron works:

1. Inputs:Similar to the McCulloch-Pitts neuron, a single-layer perceptron receives inputs (features) from the external world. Each input is associated with a weight, which represents the strength of the connection between the input and the neuron.

2. Weights: The weights are multiplied by their corresponding inputs, and the resulting weighted inputs are summed up. Mathematically, this can be represented as: Weighted Sum = Σ(w_i * x_i)

3. Bias:In addition to weights and inputs, a bias term is often included. The bias is a constant value that is added to the weighted sum before applying the activation function. It allows the network to control the threshold at which the neuron fires.

4. Activation Function:The weighted sum (including the bias) is then passed through an activation function. Common activation functions include the step function (similar to McCulloch-Pitts threshold), sigmoid function, or ReLU (Rectified Linear Unit). The activation function determines the output of the neuron based on the computed value. Output = Activation_Function(Weighted Sum + Bias)

5. Output: The final output of the single-layer perceptron is the result of applying the activation function to the weighted sum of inputs.

AND Gate

In [19]:
```
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 w
        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_thres

    # 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
```

OR Gate

In [20]:
```
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 w
        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 OR gate
    or_gate_weights = [1, 1]
    or_gate_threshold = 1

    # Create a McCulloch-Pitts neuron for the OR gate
    or_gate_neuron = McCullochPittsNeuron(or_gate_weights, or_gate_threshol

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

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

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

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

NAND Gate

In [21]:
```
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 w
        result = sum([inputs[i] * self.weights[i] for i in range(len(inputs
        return 0 if result >= self.threshold else 1

def main():
    # Define the weights and threshold for the NAND gate
    nand_gate_weights = [1, 1]
    nand_gate_threshold = 2

    # Create a McCulloch-Pitts neuron for the NAND gate
    nand_gate_neuron = McCullochPittsNeuron(nand_gate_weights, nand_gate_th

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

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

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

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

NOR Gate

In [22]:
```
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 w
        result = sum([inputs[i] * self.weights[i] for i in range(len(inputs
        return 0 if result >= self.threshold else 1

def main():
    # Define the weights and threshold for the NOR gate
    nor_gate_weights = [1, 1]
    nor_gate_threshold = 1

    # Create a McCulloch-Pitts neuron for the NOR gate
    nor_gate_neuron = McCullochPittsNeuron(nor_gate_weights, nor_gate_thres

    # Test the NOR gate
    inputs = [
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ]

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

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

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

OR Gate

In [23]:
```
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 9 epochs.
Testing the perceptron:
[0 0] -> 0
[0 1] -> 1
[1 0] -> 1
[1 1] -> 1
```

AND Gate

In [24]:
```python
def main():
    # Sample training data for AND gate
    training_data = [
        (np.array([0, 0]), 0),
        (np.array([0, 1]), 0),
        (np.array([1, 0]), 0),
        (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 4 epochs.
Testing the perceptron:
[0 0] -> 0
[0 1] -> 0
[1 0] -> 0
[1 1] -> 1
```

NAND Gate

In [25]:
```
def main():
    # Sample training data for NAND gate
    training_data = [
        (np.array([0, 0]), 1),
        (np.array([0, 1]), 1),
        (np.array([1, 0]), 1),
        (np.array([1, 1]), 0)
    ]

    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 7 epochs.
Testing the perceptron:
[0 0] -> 1
[0 1] -> 1
[1 0] -> 1
[1 1] -> 0
```

NOR Gate

In [26]:
```
def main():
    # Sample training data for NOR gate
    training_data = [
        (np.array([0, 0]), 1),
        (np.array([0, 1]), 0),
        (np.array([1, 0]), 0),
        (np.array([1, 1]), 0)
    ]

    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 11 epochs.
Testing the perceptron:
[0 0] -> 1
[0 1] -> 0
[1 0] -> 0
[1 1] -> 0
```

Conclusion: Successful Implement McCulloch Pitts neural network using Tensorflow.