Experiment - 4 McCulloch-Pitts Neural Network Model

Code

```
import numpy as np
import matplotlib.pyplot as plt

class McCullochPittsNeuron:
    def __init__(self, weights, threshold):
        """
        Initialize a McCulloch-Pitts neuron

        Parameters:
        weights (list): The weights for each input
        threshold (float): The threshold for activation
        """
        self.weights = np.array(weights)
        self.threshold = threshold

    def activate(self, inputs):
        """
        Calculate the output of the neuron given the inputs

        Parameters:
        inputs (list): The input values

        Returns:
        int: 1 if the weighted sum is greater than or equal to threshold, 0
otherwise
        """
        # Calculate the weighted sum
        weighted_sum = np.dot(inputs, self.weights)

        # Apply the threshold activation function
        return 1 if weighted_sum >= self.threshold else 0

    def test_logic_gate(self, inputs_matrix):
        """
        Test the neuron on multiple input combinations

        Parameters:
        inputs_matrix (list of lists): Matrix containing input combinations

        Returns:
        list: The outputs for each input combination
        """
        outputs = []
        for inputs in inputs_matrix:
            output = self.activate(inputs)
            outputs.append(output)
```

```
            print(f"Inputs: {inputs}, Output: {output}")
        return outputs

def plot_logic_gate(gate_name, inputs, outputs):
    """
    Plot the truth table of a logic gate

    Parameters:
    gate_name (str): The name of the logic gate
    inputs (list of lists): Matrix containing input combinations
    outputs (list): The outputs for each input combination
    """
    plt.figure(figsize=(8, 6))

    # For 2 input gates
    if len(inputs[0]) == 2:
        x1 = [row[0] for row in inputs]
        x2 = [row[1] for row in inputs]
        plt.scatter(x1, x2, c=outputs, cmap='coolwarm', s=200, alpha=0.8)

        for i, (x1_val, x2_val) in enumerate(zip(x1, x2)):
            plt.annotate(f"Output: {outputs[i]}",
                         (x1_val, x2_val),
                         xytext=(10, 5),
                         textcoords='offset points')

        plt.xlabel('Input 1')
        plt.ylabel('Input 2')
        plt.grid(True)
        plt.title(f'{gate_name} Gate')
        plt.savefig(f"{gate_name.lower()}_gate.png")

    # For single input gates (like NOT)
    elif len(inputs[0]) == 1:
        x = [row[0] for row in inputs]
        plt.scatter(x, outputs, c=outputs, cmap='coolwarm', s=200, alpha=0.8)

        for i, x_val in enumerate(x):
            plt.annotate(f"Input: {x_val}, Output: {outputs[i]}",
                         (x_val, outputs[i]),
                         xytext=(10, 5),
                         textcoords='offset points')

        plt.xlabel('Input')
        plt.ylabel('Output')
        plt.grid(True)
        plt.title(f'{gate_name} Gate')
        plt.savefig(f"{gate_name.lower()}_gate.png")

def main():
    # Define input combinations for 2-input logic gates
    inputs_2bit = [
        [0, 0],
```

```
        [0, 1],
        [1, 0],
        [1, 1]
    ]

    print("\n===== AND Gate =====")
    # AND gate: both inputs must be 1 to get output 1
    and_neuron = McCullochPittsNeuron(weights=[1, 1], threshold=2)
    and_outputs = and_neuron.test_logic_gate(inputs_2bit)
    plot_logic_gate("AND", inputs_2bit, and_outputs)

    print("\n===== OR Gate =====")
    # OR gate: at least one input must be 1 to get output 1
    or_neuron = McCullochPittsNeuron(weights=[1, 1], threshold=1)
    or_outputs = or_neuron.test_logic_gate(inputs_2bit)
    plot_logic_gate("OR", inputs_2bit, or_outputs)

    print("\n===== NAND Gate =====")
    # NAND gate: only when both inputs are 1, output is 0
    nand_neuron = McCullochPittsNeuron(weights=[-1, -1], threshold=-1)
    nand_outputs = nand_neuron.test_logic_gate(inputs_2bit)
    plot_logic_gate("NAND", inputs_2bit, nand_outputs)

    print("\n===== NOR Gate =====")
    # NOR gate: only when both inputs are 0, output is 1
    nor_neuron = McCullochPittsNeuron(weights=[-1, -1], threshold=0)
    nor_outputs = nor_neuron.test_logic_gate(inputs_2bit)
    plot_logic_gate("NOR", inputs_2bit, nor_outputs)

    # Define input combinations for NOT gate
    inputs_1bit = [
        [0],
        [1]
    ]

    print("\n===== NOT Gate =====")
    # NOT gate: invert the input
    not_neuron = McCullochPittsNeuron(weights=[-1], threshold=0)
    not_outputs = not_neuron.test_logic_gate(inputs_1bit)
    plot_logic_gate("NOT", inputs_1bit, not_outputs)

    # XOR gate cannot be implemented with a single McCulloch-Pitts neuron
    # It requires a network of neurons
    print("\n===== XOR Gate =====")
    print("XOR cannot be implemented with a single McCulloch-Pitts neuron.")
    print("It requires a network of neurons.")

    plt.show()

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

Output

```
===== AND Gate =====
Inputs: [0, 0], Output: 0
Inputs: [0, 1], Output: 0
Inputs: [1, 0], Output: 0
Inputs: [1, 1], Output: 1

===== OR Gate =====
Inputs: [0, 0], Output: 0
Inputs: [0, 1], Output: 1
Inputs: [1, 0], Output: 1
Inputs: [1, 1], Output: 1

===== NAND Gate =====
Inputs: [0, 0], Output: 1
Inputs: [0, 1], Output: 1
Inputs: [1, 0], Output: 1
Inputs: [1, 1], Output: 0

===== NOR Gate =====
Inputs: [0, 0], Output: 1
Inputs: [0, 1], Output: 0
Inputs: [1, 0], Output: 0
Inputs: [1, 1], Output: 0

===== NOT Gate =====
Inputs: [0], Output: 1
Inputs: [1], Output: 0
```