LAB EXERCISE 1

You have been tasked with building simple neural networks to simulate the behavior of logic gates using a Single Layer Perceptron. This task will involve constructing, training, and testing perceptrons for the following gates: AND, OR, AND-NOT and XOR.

1. AND Gate Classification
   • Scenario:
   You are tasked with building a simple neural network to simulate an AND gate using a Single Layer Perceptron. The AND gate outputs 1 only if both inputs are 1; otherwise, it outputs 0.
   • Lab Task: Implement a Single Layer Perceptron using Python in Google Colab to classify the output of an AND gate given two binary inputs (0 or 1).
   Follow these steps:
   • Create a dataset representing the truth table of the AND gate.
   • Define the perceptron model with one neuron, including the activation function and weights initialization(Try both random weights and defined weights).
   • Train the perceptron using a suitable learning algorithm (e.g., gradient descent).
   • Test the model with all possible input combinations and display the results.
   Questions:
   • How do the weights and bias values change during training for the AND gate?
   • Can the perceptron successfully learn the AND logic with a linear decision boundary?

2. OR Gate Classification
   • Scenario:
   Your next task is to design a perceptron that mimics the behavior of an OR gate. The OR gate outputs 1 if at least one of its inputs is 1.
   • Lab Task: Using Google Colab, create a Single Layer Perceptron to classify the output of an OR gate.
   Perform the following steps:
   • Prepare the dataset for the OR gate's truth table.
   • Define and initialize a Single Layer Perceptron model.
   • Implement the training process and adjust the perceptron's weights.
   • Validate the perceptron's performance with the OR gate input combinations.
   Questions:
   • What changes in the perceptron's weights are necessary to represent the OR gate logic?

• How does the linear decision boundary look for the OR gate classification?

3. AND-NOT Gate Classification
   • Scenario:
   You need to implement an AND-NOT gate, which outputs 1 only if the first input is 1 and the second input is 0.
   • Lab Task: Design a Single Layer Perceptron in Google Colab to classify the output of an AND-NOT gate.
   Follow these steps:
   • Create the truth table for the AND-NOT gate.
   • Define a perceptron model with an appropriate activation function.
   • Train the model on the AND-NOT gate dataset.
   • Test the model and analyze its classification accuracy.
   Questions:
   • What is the perceptron's weight configuration after training for the AND-NOT gate?
   • How does the perceptron handle cases where both inputs are 1 or 0?

4. XOR Gate Classification
   • Scenario:
   The XOR gate is known for its complexity, as it outputs 1 only when the inputs are different. This is a challenge for a Single Layer Perceptron since XOR is not linearly separable.
   • Lab Task: Attempt to implement a Single Layer Perceptron in Google Colab to classify the output of an XOR gate.
   Perform the following steps:
   • Create the XOR gate's truth table dataset.
   • Implement the perceptron model and train it using the XOR dataset.
   • Observe and discuss the perceptron's performance in this scenario.
   Questions:
   • Why does the Single Layer Perceptron struggle to classify the XOR gate?
   • What modifications can be made to the neural network model to handle the XOR gate correctly?

   Instructions for Lab Work in Google Colab:Setup: Import necessary libraries (e.g., NumPy) and create datasets for each logic gate.
   Model Implementation: Write the code for the Single Layer Perceptron, including forward and backward passes.
   Training: Use a learning algorithm to adjust weights and biases.
   Evaluation: Test the perceptron's performance for each logic gate and plot the decision boundaries where applicable.
   Analysis: Document the results and answer the provided questions.

AND GATE CLASSIFICATION

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

In [1]:
```python
import numpy as np

# AND gate truth table
X_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Inputs
y_and = np.array([0, 0, 0, 1])  # Output (AND logic)
```

In [15]:
```python
#Function for perceptron
# Step 1: Define the activation function (Heaviside Step function)
def activation_function(x):
    return 1 if x >= 0 else 0

# Step 2: Define the perceptron model
class Perceptron:
    def __init__(self, learning_rate=0.1):
        # Initialize weights randomly (You can also set custom weights for c
        self.weights = np.random.rand(2)  # Two weights for two inputs
        self.bias = np.random.rand()      # Bias term
        self.learning_rate = learning_rate

    # Step 3: Perceptron prediction (forward pass)
    def predict(self, inputs):
        # Calculate the weighted sum of inputs + bias
        weighted_sum = np.dot(inputs, self.weights) + self.bias
        # Apply the activation function
        return activation_function(weighted_sum)

    # Step 4: Train the perceptron using the training data
    def train(self, training_inputs, labels, epochs=20):
        for epoch in range(epochs):
            print(f'Epoch {epoch+1}/{epochs}:')
            for inputs, label in zip(training_inputs, labels):
                # Make a prediction
                prediction = self.predict(inputs)
                # Calculate the error (difference between predicted and actu
                error = label - prediction
                # Update the weights and bias using the perceptron learning
                self.weights += self.learning_rate * error * inputs
                self.bias += self.learning_rate * error
                print(f' Inputs: {inputs}, Prediction: {prediction}, Actual:
                print(f' Updated Weights: {self.weights}, Updated Bias: {sel
            print()
```

```
# Step 6: Initialize the perceptron model
perceptron = Perceptron(learning_rate=0.1)
```

In [16]:

```
# Step 7: Train the perceptron model
perceptron.train(X_and, y_and, epochs=10)

# Step 8: Test the perceptron model
print('Testing the model:')
for inputs in X_and:
    output = perceptron.predict(inputs)
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.38419313 0.42441188], Updated Bias: 0.8022491958132121
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.38419313 0.32441188], Updated Bias: 0.7022491958132121
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.28419313 0.32441188], Updated Bias: 0.6022491958132121
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.28419313 0.32441188], Updated Bias: 0.6022491958132121

Epoch 2/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.28419313 0.32441188], Updated Bias: 0.5022491958132121
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.28419313 0.22441188], Updated Bias: 0.40224919581321217
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.18419313 0.22441188], Updated Bias: 0.3022491958132122
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.22441188], Updated Bias: 0.3022491958132122

Epoch 3/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.18419313 0.22441188], Updated Bias: 0.20224919581321218
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.18419313 0.12441188], Updated Bias: 0.10224919581321218
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.08419313 0.12441188], Updated Bias: 0.00224919581321217
37
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.08419313 0.12441188], Updated Bias: 0.00224919581321217
37

Epoch 4/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.08419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.08419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.08419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
```

```
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
```

```
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4

Testing the model:
 Inputs: [0 0], Predicted Output: 0
 Inputs: [0 1], Predicted Output: 0
 Inputs: [1 0], Predicted Output: 0
 Inputs: [1 1], Predicted Output: 1
```

Weight and Bias Changes: The weights and bias are adjusted based on the error between the predicted and actual outputs during each training iteration.
Linear Decision Boundary: Yes, the perceptron can successfully learn the AND gate logic as it is linearly separable.

OR GATE CLASSIFICATION

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

In [5]:
```
# OR gate truth table
X_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_or = np.array([0, 1, 1, 1])
```

In [17]:
```
perceptron.train(X_or, y_or, epochs=10)

print('Testing the model:')
for inputs in X_or:
    output = perceptron.predict(inputs)
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 2/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 3/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 4/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
```

```
Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
```

```
Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3

Testing the model:
 Inputs: [0 0], Predicted Output: 0
 Inputs: [0 1], Predicted Output: 1
 Inputs: [1 0], Predicted Output: 1
 Inputs: [1 1], Predicted Output: 1
```

Weight Changes: The weights will adjust to represent the OR logic, which requires a different decision boundary from AND.

Linear Decision Boundary: The OR logic can be learned with a linear decision boundary because it's also linearly separable.

AND-NOT CLASSIFICATION

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	1
1	1	0

In [8]:
```
X_andnot = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_andnot = np.array([0, 0, 1, 0])
```

In [18]:

```
perceptron.train(X_andnot, y_andnot, epochs=10)

print('Testing the model:')
for inputs in X_andnot:
    output = perceptron.predict(inputs)
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.18419313 0.12441188], Updated Bias: -0.0977508041867878
3
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.18419313 0.02441188], Updated Bias: -0.1977508041867878
4
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.28419313 0.02441188], Updated Bias: -0.0977508041867878
3
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.18419313 -0.07558812], Updated Bias: -0.19775080418678
784

Epoch 2/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.07558812], Updated Bias: -0.19775080418678
784
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.07558812], Updated Bias: -0.19775080418678
784
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.28419313 -0.07558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.18419313 -0.17558812], Updated Bias: -0.19775080418678
784

Epoch 3/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.17558812], Updated Bias: -0.19775080418678
784
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.17558812], Updated Bias: -0.19775080418678
784
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.28419313 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.18419313 -0.27558812], Updated Bias: -0.19775080418678
784

Epoch 4/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.27558812], Updated Bias: -0.19775080418678
784
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.27558812], Updated Bias: -0.19775080418678
784
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
```

```
Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
```

```
Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783

Testing the model:
 Inputs: [0 0], Predicted Output: 0
 Inputs: [0 1], Predicted Output: 0
 Inputs: [1 0], Predicted Output: 1
 Inputs: [1 1], Predicted Output: 0
```

Weight Configuration: The weights will configure to output 1 only when the first input is 1 and the second is 0.

Handling Special Cases: The perceptron will output 0 when both inputs are 1 or both are 0, correctly classifying the AND-NOT logic.

XOR GATE CLASSIFICATION

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

In [12]:
```
# XOR gate truth table
X_xor = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_xor = np.array([0, 1, 1, 0])
```

In [19]:

```
perceptron.train(X_xor, y_xor, epochs=10)

print('Testing the model:')
for inputs in X_xor:
    output = perceptron.predict(inputs)
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.28419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.28419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.28419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.18419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 2/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.18419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.18419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.18419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.08419313 -0.27558812], Updated Bias: -0.09775080418678
783

Epoch 3/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.08419313 -0.27558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.08419313 -0.17558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.08419313 -0.17558812], Updated Bias: 0.002249195813212
1737

Epoch 4/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.08419313 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
```

```
Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737

Epoch 6/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737

Epoch 7/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737

Epoch 8/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737
```

```
Epoch 9/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737

Epoch 10/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: -0.09775080418678
783
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.01580687 -0.07558812], Updated Bias: 0.002249195813212
1737
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [ 0.08419313 -0.07558812], Updated Bias: 0.102249195813212
18
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.01580687 -0.17558812], Updated Bias: 0.002249195813212
1737

Testing the model:
 Inputs: [0 0], Predicted Output: 1
 Inputs: [0 1], Predicted Output: 0
 Inputs: [1 0], Predicted Output: 0
 Inputs: [1 1], Predicted Output: 0
```

Training: The XOR logic is not linearly separable, so a Single Layer Perceptron will struggle to learn it.

Struggle with XOR: The XOR gate requires a non-linear decision boundary, which a Single Layer Perceptron cannot represent due to its linear nature.

Modifications for XOR: To handle XOR, you would need a Multi-Layer Perceptron (MLP) with at least one hidden layer, enabling the model to capture non-linear patterns.