2347152

September 20, 2024

Inputs	AND
0, 0	0
0, 1	0
1, 0	0
1, 1	1

[13]:
```python
import numpy as np

# Step 1: Define the activation function (Heaviside Step function)
# This function decides if the neuron should "fire" (output 1) or not (output 0)
def activation_function(x):
    return 1 if x >= 0 else 0  # Return 1 if x is greater than or equal to 0, otherwise return 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 demonstration)
        self.weights = np.random.rand(2)  # Two weights for two inputs (e.g., x1 and x2)
        self.bias = np.random.rand()      # Bias term, helps adjust the output
        self.learning_rate = learning_rate  # The rate at which the model learns

    # 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 to determine the output
        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}:')  # Show which epoch we are on
            for inputs, label in zip(training_inputs, labels):
```

```
                # Make a prediction using the current weights and bias
                prediction = self.predict(inputs)
                # Calculate the error (how far off the prediction is from the␣
 ↪actual label)
                error = label - prediction
                # Update the weights and bias using the perceptron learning rule
                self.weights += self.learning_rate * error * inputs  # Adjust␣
 ↪weights
                self.bias += self.learning_rate * error  # Adjust bias
                # Print details of the current step
                print(f' Inputs: {inputs}, Prediction: {prediction}, Actual:␣
 ↪{label}, Error: {error}')
                print(f' Updated Weights: {self.weights}, Updated Bias: {self.
 ↪bias}')
            print()  # Print a new line for clarity after each epoch

# Step 5: Prepare the dataset for the AND gate (Truth Table)
# The training inputs represent the combinations of inputs for the AND gate
training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
# The labels are the expected outputs for the AND gate
labels = np.array([0, 0, 0, 1])  # AND gate outputs: 0, 0, 0, 1

# Step 6: Initialize the Perceptron model
perceptron = Perceptron(learning_rate=0.1)  # Create an instance of the␣
 ↪Perceptron

# Step 7: Train the perceptron model using the training data
perceptron.train(training_inputs, labels, epochs=10)

# Step 8: Test the perceptron model after training
print('Testing the model:')
for inputs in training_inputs:
    output = perceptron.predict(inputs)  # Get the model's output for each input
    print(f' Inputs: {inputs}, Predicted Output: {output}')  # Display the␣
 ↪result
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.14911035 0.4145114 ], Updated Bias: 0.7906927418670363
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.14911035 0.3145114 ], Updated Bias: 0.6906927418670363
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.04911035 0.3145114 ], Updated Bias: 0.5906927418670364
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.04911035 0.3145114 ], Updated Bias: 0.5906927418670364

Epoch 2/10:
```

```
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.04911035 0.3145114 ], Updated Bias: 0.4906927418670364
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.04911035 0.2145114 ], Updated Bias: 0.3906927418670364
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.05088965  0.2145114 ], Updated Bias: 0.2906927418670364
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [-0.05088965  0.2145114 ], Updated Bias: 0.2906927418670364

Epoch 3/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.05088965  0.2145114 ], Updated Bias: 0.1906927418670364
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.05088965  0.1145114 ], Updated Bias: 0.09069274186703641
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.15088965  0.1145114 ], Updated Bias: -0.009307258132963597
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.05088965  0.2145114 ], Updated Bias: 0.09069274186703641

Epoch 4/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.05088965  0.2145114 ], Updated Bias: -0.009307258132963597
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.05088965  0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [-0.05088965  0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.04911035 0.2145114 ], Updated Bias: -0.009307258132963597

Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.04911035 0.2145114 ], Updated Bias: -0.009307258132963597
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.04911035 0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.04911035 0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.04911035 0.1145114 ], Updated Bias: -0.1093072581329636

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.04911035 0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.04911035 0.0145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.04911035 0.0145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.14911035 0.1145114 ], Updated Bias: -0.1093072581329636
```

```
Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.14911035 0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.14911035 0.0145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.14911035 0.0145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.1093072581329636

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.1093072581329636
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.24911035 0.0145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.14911035 0.0145114 ], Updated Bias: -0.3093072581329636
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.2093072581329636

Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.2093072581329636
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.2093072581329636
 Inputs: [1 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.14911035 0.1145114 ], Updated Bias: -0.3093072581329636
 Inputs: [1 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.24911035 0.2145114 ], Updated Bias: -0.2093072581329636

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.24911035 0.2145114 ], Updated Bias: -0.2093072581329636
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.3093072581329636
 Inputs: [1 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.3093072581329636
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.24911035 0.1145114 ], Updated Bias: -0.3093072581329636

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
```

0.0.1 1. Weight and Bias Changes During Training:

Weights and bias are updated based on the error between the predicted and actual output. The update rule is: [weights$\{new\}$ = $weights\{\text{old}\}$ + learning rate ×error ×input] [bias$\{new\}$ = $bias\{\text{old}\}$ + learning rate ×error] As training progresses, the weights and bias adjust to minimize the error, ensuring correct predictions for the AND gate truth table.

0.0.2 2. Learning AND Logic with a Linear Decision Boundary:

Yes, the perceptron can successfully learn the AND logic because the AND gate is linearly separable. The perceptron finds weights and bias to define a linear boundary that separates the output classes (0 and 1) correctly.

Inputs	OR
0, 0	0
0, 1	1
1, 0	1
1, 1	1

[7]:
```
# Step 5: Prepare the dataset for the OR gate (Truth Table)
# The training inputs represent the combinations of inputs for the OR gate
training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
# The labels are the expected outputs for the OR gate
# The OR gate outputs: 0 if both inputs are 0, and 1 if at least one input is 1
labels = np.array([0, 1, 1, 1])  # OR gate outputs: 0, 1, 1, 1

# Step 6: Initialize the Perceptron model
# Create an instance of the Perceptron with a specified learning rate
perceptron = Perceptron(learning_rate=0.1)

# Step 7: Train the perceptron model using the training data
# The model will learn from the training inputs and labels over multiple epochs
perceptron.train(training_inputs, labels, epochs=10)

# Step 8: Test the perceptron model after training
print('Testing the model:')  # Indicate we are starting the testing phase
# Loop through each set of inputs to see how well the model performs
for inputs in training_inputs:
    output = perceptron.predict(inputs)  # Get the model's predicted output for␣
 ↪the current inputs
    # Print the inputs and the model's predicted output
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.2592929  0.67005697], Updated Bias: 0.09501909878104661
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
```

```
 Updated Weights: [0.2592929  0.67005697], Updated Bias: 0.09501909878104661
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: 0.09501909878104661
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: 0.09501909878104661

Epoch 2/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 3/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 4/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
```

```
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
 Inputs: [1 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.2592929  0.67005697], Updated Bias: -0.004980901218953393
```

```
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
```

0.0.3 1. Necessary Weight Changes for OR Gate Logic:

The perceptron's weights and bias must adjust to ensure that it outputs **1** when at least one input is 1. Weights increase when the prediction is incorrect, encouraging the model to correctly classify `(0, 1)`, `(1, 0)`, and `(1, 1)` as 1.

0.0.4 2. Linear Decision Boundary for OR Gate:

The linear decision boundary is a straight line that separates the input `(0, 0)` (output 0) from the inputs `(0, 1)`, `(1, 0)`, and `(1, 1)` (output 1). This boundary ensures correct classification for the OR gate.

Inputs	AND-NOT
0, 0	0
0, 1	0
1, 0	1
1, 1	0

[8]:
```python
# Step 1: Prepare the truth table for the AND-NOT gate
# The AND-NOT gate outputs 1 only when the first input is 1 and the second input is 0
# The training inputs represent all combinations of two binary inputs
training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
# The labels are the expected outputs for the AND-NOT gate based on the inputs
# Expected outputs:
# - (0, 0) => 0 (because the first input is 0)
# - (0, 1) => 0 (because the first input is 0)
# - (1, 0) => 1 (because the first input is 1 and second input is 0)
# - (1, 1) => 0 (because the second input is 1)
labels = np.array([0, 0, 1, 0])  # AND-NOT gate outputs

# Step 2: Initialize the perceptron model
# Create an instance of the Perceptron with a specified learning rate
perceptron = Perceptron(learning_rate=0.1)

# Step 3: Train the perceptron on the AND-NOT gate truth table
# The model will learn from the training inputs and their corresponding labels
perceptron.train(training_inputs, labels, epochs=10)

# Step 4: Test the perceptron model after training
print('Testing the model:')  # Indicate we are starting the testing phase
```

```
# Loop through each set of inputs to evaluate the model's performance
for inputs in training_inputs:
    output = perceptron.predict(inputs)  # Get the model's predicted output for␣
 ↪the current inputs
    # Print the inputs and the model's predicted output
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.91886217 0.34781154], Updated Bias: 0.7057034600668574
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.91886217 0.24781154], Updated Bias: 0.6057034600668574
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.91886217 0.24781154], Updated Bias: 0.6057034600668574
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.81886217 0.14781154], Updated Bias: 0.5057034600668574

Epoch 2/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.81886217 0.14781154], Updated Bias: 0.4057034600668574
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.81886217 0.04781154], Updated Bias: 0.30570346006685745
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.81886217 0.04781154], Updated Bias: 0.30570346006685745
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.71886217 -0.05218846], Updated Bias: 0.20570346006685744

Epoch 3/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.71886217 -0.05218846], Updated Bias: 0.10570346006685744
 Inputs: [0 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.71886217 -0.15218846], Updated Bias: 0.00570346006685743
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.71886217 -0.15218846], Updated Bias: 0.00570346006685743
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.61886217 -0.25218846], Updated Bias: -0.09429653993314258

Epoch 4/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.61886217 -0.25218846], Updated Bias: -0.09429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.61886217 -0.25218846], Updated Bias: -0.09429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.61886217 -0.25218846], Updated Bias: -0.09429653993314258
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
```

```
Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
```

```
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [0 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258
 Inputs: [1 1], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.51886217 -0.35218846], Updated Bias: -0.19429653993314258

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
```

0.0.5 1. Perceptron's Weight Configuration After Training for AND-NOT Gate:

After training, the weights will be adjusted so that the perceptron outputs **1** when the first input is 1 and the second input is 0. Typically, the weight for the first input will be positive, while the weight for the second input will be negative, ensuring correct classification.

0.0.6 2. Handling Cases Where Both Inputs Are 1 or 0:

- **Both inputs are 1**: The perceptron outputs **0** because the second input's negative weight cancels the first input's positive effect.
- **Both inputs are 0**: The perceptron outputs **0** as the weighted sum of inputs will be below the activation threshold.

Inputs	XOR
0, 0	0
0, 1	1
1, 0	1
1, 1	0

[14]:
```
# Step 1: Prepare the truth table for the XOR gate
# The XOR gate (exclusive OR) outputs 1 when the inputs are different (one is␣
 ↪1, the other is 0)
# The training inputs represent all combinations of two binary inputs
training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
# The labels are the expected outputs for the XOR gate based on the inputs
# Expected outputs:
# - (0, 0) => 0 (both inputs are 0)
# - (0, 1) => 1 (one input is 1 and the other is 0)
# - (1, 0) => 1 (one input is 1 and the other is 0)
```

```python
# - (1, 1) => 0 (both inputs are 1)
labels = np.array([0, 1, 1, 0])  # XOR gate outputs

# Step 2: Initialize the perceptron model
# Create an instance of the Perceptron with a specified learning rate
perceptron = Perceptron(learning_rate=0.1)

# Step 3: Train the perceptron on the XOR gate truth table
# The model will learn from the training inputs and their corresponding labels
perceptron.train(training_inputs, labels, epochs=10)

# Step 4: Test the perceptron model after training
print('Testing the model:')  # Indicate we are starting the testing phase
# Loop through each set of inputs to evaluate the model's performance
for inputs in training_inputs:
    output = perceptron.predict(inputs)  # Get the model's predicted output for␣
 ↪the current inputs
    # Print the inputs and the model's predicted output
    print(f' Inputs: {inputs}, Predicted Output: {output}')
```

```
Epoch 1/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.40414398 0.67899573], Updated Bias: 0.42459388166132306
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.40414398 0.67899573], Updated Bias: 0.42459388166132306
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.40414398 0.67899573], Updated Bias: 0.42459388166132306
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.30414398 0.57899573], Updated Bias: 0.3245938816613231

Epoch 2/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.30414398 0.57899573], Updated Bias: 0.22459388166132307
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.30414398 0.57899573], Updated Bias: 0.22459388166132307
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.30414398 0.57899573], Updated Bias: 0.22459388166132307
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.20414398 0.47899573], Updated Bias: 0.12459388166132307

Epoch 3/10:
 Inputs: [0 0], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.20414398 0.47899573], Updated Bias: 0.024593881661323064
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.20414398 0.47899573], Updated Bias: 0.024593881661323064
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.20414398 0.47899573], Updated Bias: 0.024593881661323064
```

```
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.10414398 0.37899573], Updated Bias: -0.07540611833867694

Epoch 4/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.10414398 0.37899573], Updated Bias: -0.07540611833867694
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.10414398 0.37899573], Updated Bias: -0.07540611833867694
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.10414398 0.37899573], Updated Bias: -0.07540611833867694
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.00414398 0.27899573], Updated Bias: -0.17540611833867695

Epoch 5/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.00414398 0.27899573], Updated Bias: -0.17540611833867695
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.00414398 0.27899573], Updated Bias: -0.17540611833867695
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.10414398 0.27899573], Updated Bias: -0.07540611833867694
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.00414398 0.17899573], Updated Bias: -0.17540611833867695

Epoch 6/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.00414398 0.17899573], Updated Bias: -0.17540611833867695
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.00414398 0.17899573], Updated Bias: -0.17540611833867695
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.10414398 0.17899573], Updated Bias: -0.07540611833867694
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.00414398 0.07899573], Updated Bias: -0.17540611833867695

Epoch 7/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.00414398 0.07899573], Updated Bias: -0.17540611833867695
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.00414398 0.17899573], Updated Bias: -0.07540611833867694
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.10414398 0.17899573], Updated Bias: 0.024593881661323064
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [0.00414398 0.07899573], Updated Bias: -0.07540611833867694

Epoch 8/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [0.00414398 0.07899573], Updated Bias: -0.07540611833867694
 Inputs: [0 1], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.00414398 0.07899573], Updated Bias: -0.07540611833867694
```

```
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.10414398 0.07899573], Updated Bias: 0.024593881661323064
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [ 0.00414398 -0.02100427], Updated Bias: -0.07540611833867694

Epoch 9/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [ 0.00414398 -0.02100427], Updated Bias: -0.07540611833867694
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.00414398 0.07899573], Updated Bias: 0.024593881661323064
 Inputs: [1 0], Prediction: 1, Actual: 1, Error: 0
 Updated Weights: [0.00414398 0.07899573], Updated Bias: 0.024593881661323064
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.09585602 -0.02100427], Updated Bias: -0.07540611833867694

Epoch 10/10:
 Inputs: [0 0], Prediction: 0, Actual: 0, Error: 0
 Updated Weights: [-0.09585602 -0.02100427], Updated Bias: -0.07540611833867694
 Inputs: [0 1], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [-0.09585602  0.07899573], Updated Bias: 0.024593881661323064
 Inputs: [1 0], Prediction: 0, Actual: 1, Error: 1
 Updated Weights: [0.00414398 0.07899573], Updated Bias: 0.12459388166132307
 Inputs: [1 1], Prediction: 1, Actual: 0, Error: -1
 Updated Weights: [-0.09585602 -0.02100427], Updated Bias: 0.024593881661323064

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

0.0.7 1. Why Does the Single Layer Perceptron Struggle to Classify the XOR Gate?

The Single Layer Perceptron struggles with the XOR gate because it is **not linearly separable**. This means that a single straight line cannot divide the input space into two classes (0 and 1). The XOR function requires a more complex decision boundary that cannot be achieved with only one layer.

0.0.8 2. Modifications to Handle the XOR Gate Correctly:

To correctly classify the XOR gate, the following modifications can be made: - **Use a Multi-Layer Perceptron (MLP)**: Introduce one or more hidden layers to allow for non-linear transformations. - **Increase the Number of Neurons**: Add more neurons in the hidden layer(s) to capture complex patterns. - **Train with Sufficient Epochs**: Increase the number of training epochs to ensure proper convergence.