LAB 1

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

Submitted by:

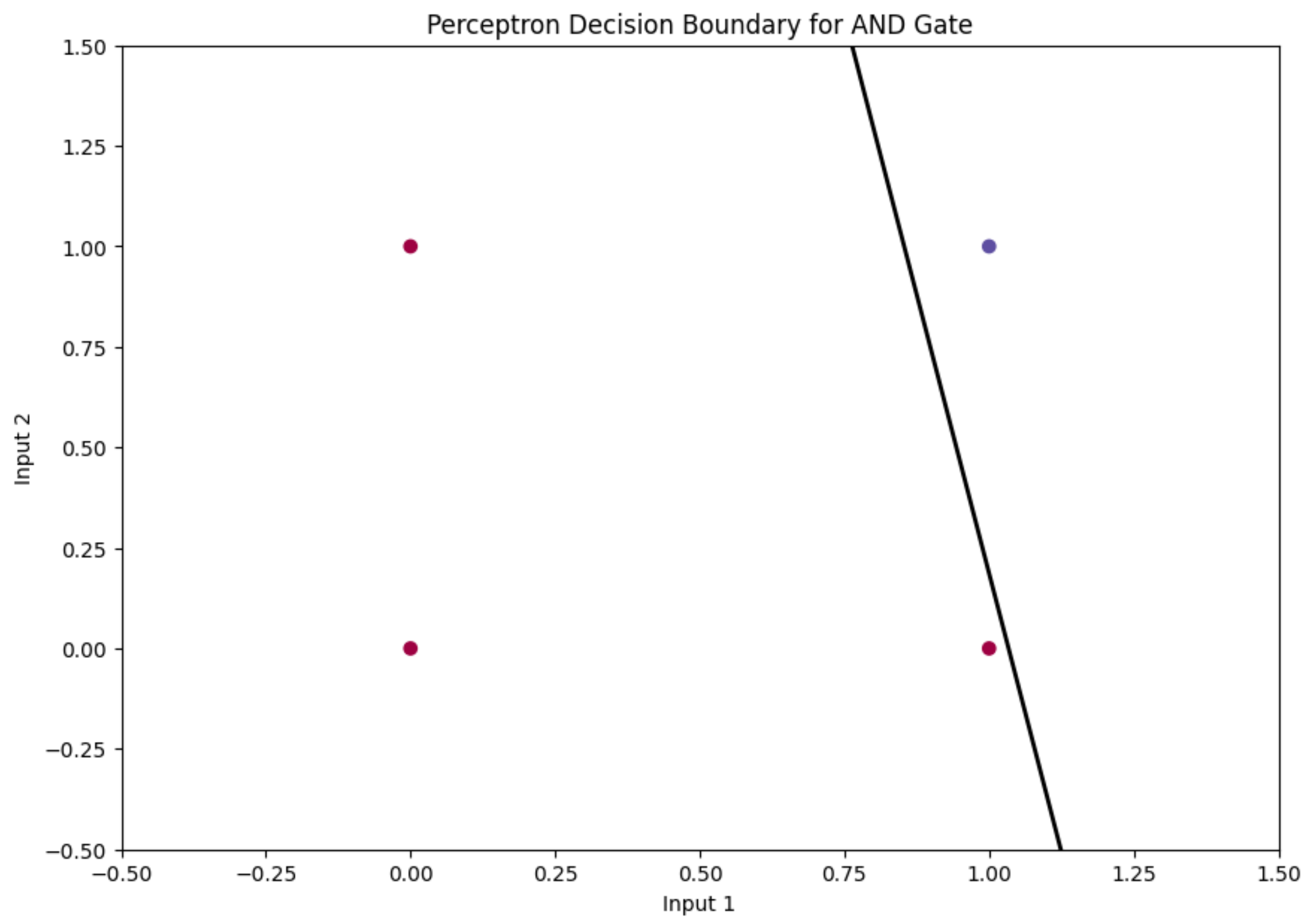
Punith Raj S P

AND Gate Classification

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

# Create the dataset (AND gate truth table)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

# Perceptron class
class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=100, random_w
        self.lr = learning_rate
        self.epochs = epochs
        if random_weights:
            self.weights = np.random.rand(input_size)
            self.bias = np.random.rand()
        else:
            self.weights = np.zeros(input_size)
            self.bias = 0

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        sum = np.dot(inputs, self.weights) + self.bias
        return self.activation_function(sum)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.weights += self.lr * (label - prediction) * inputs
                self.bias += self.lr * (label - prediction)

# Function to test the perceptron
def test_perceptron(perceptron, X, y):
    print("\nTesting the perceptron:")
    for inputs, label in zip(X, y):
        prediction = perceptron.predict(inputs)
        print(f"Inputs: {inputs}, Target: {label}, Prediction: {predictior

# Train and test with random weights
print("Training with random weights:")
perceptron_random = Perceptron(input_size=2, random_weights=True)
perceptron_random.train(X, y)
test_perceptron(perceptron_random, X, y)

# Train and test with defined (zero) weights
```

```
print("\nTraining with defined (zero) weights:")
perceptron_defined = Perceptron(input_size=2, random_weights=False)
perceptron_defined.train(X, y)
test_perceptron(perceptron_defined, X, y)

# Visualize the decision boundary
def plot_decision_boundary(perceptron, X, y):
    plt.figure(figsize=(10, 7))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

    x1 = np.linspace(-0.5, 1.5, 10)
    x2 = -(perceptron.weights[0] * x1 + perceptron.bias) / perceptron.weig

    plt.plot(x1, x2, c='k', lw=2)
    plt.xlim([-0.5, 1.5])
    plt.ylim([-0.5, 1.5])
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.title('Perceptron Decision Boundary for AND Gate')
    plt.show()

plot_decision_boundary(perceptron_random, X, y)
```

```
Training with random weights:

Testing the perceptron:
Inputs: [0 0], Target: 0, Prediction: 0
Inputs: [0 1], Target: 0, Prediction: 0
Inputs: [1 0], Target: 0, Prediction: 0
Inputs: [1 1], Target: 1, Prediction: 1

Training with defined (zero) weights:

Testing the perceptron:
Inputs: [0 0], Target: 0, Prediction: 0
Inputs: [0 1], Target: 0, Prediction: 0
Inputs: [1 0], Target: 0, Prediction: 0
Inputs: [1 1], Target: 1, Prediction: 1
```



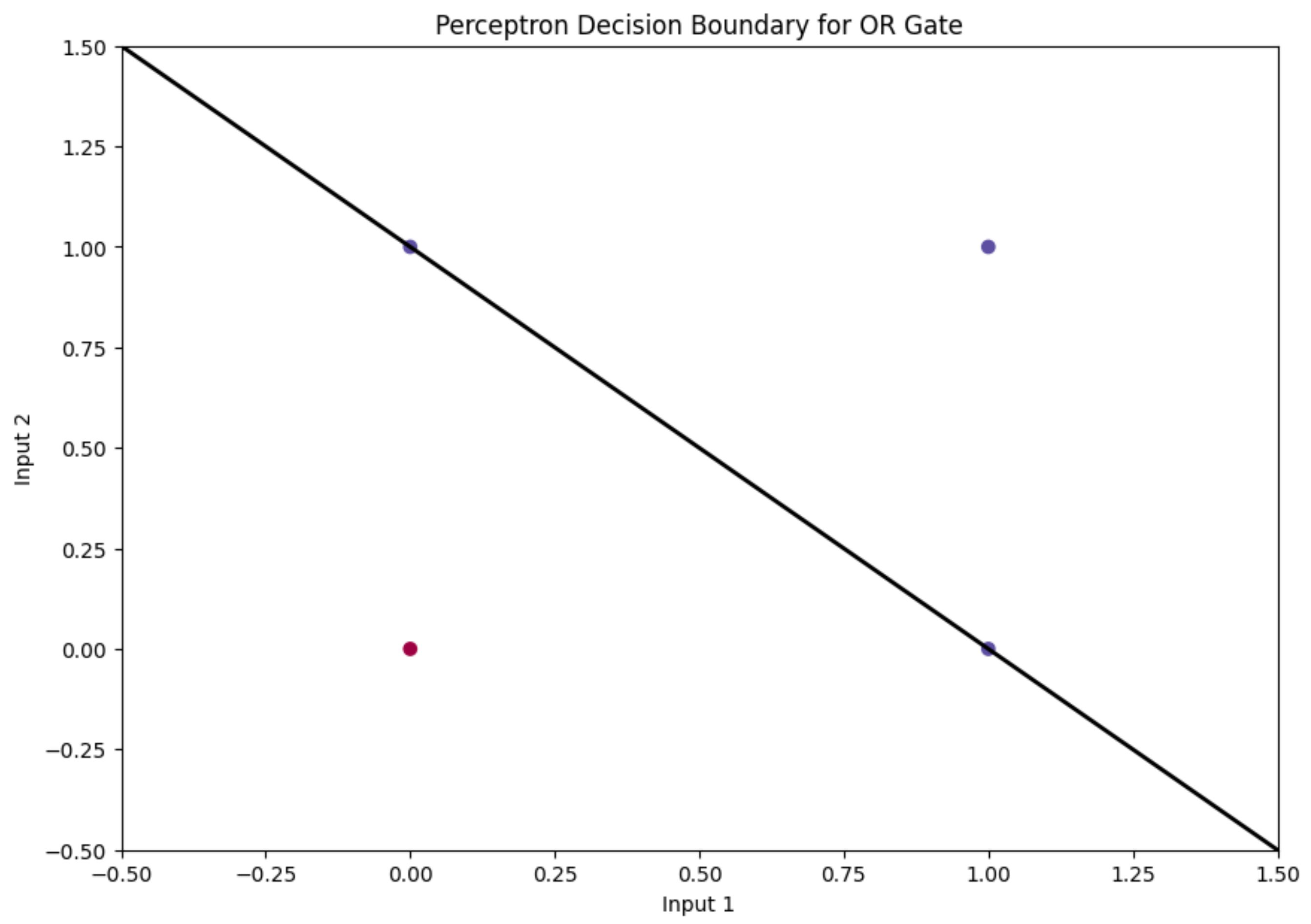
OR Gate Classification

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

# Prepare the dataset (OR gate truth table)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])

# Perceptron class
```

```
class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=100):
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = np.zeros(input_size)
        self.bias = 0

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        sum = np.dot(inputs, self.weights) + self.bias
        return self.activation_function(sum)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.weights += self.lr * (label - prediction) * inputs
                self.bias += self.lr * (label - prediction)

# Function to test the perceptron
def test_perceptron(perceptron, X, y):
    print("\nTesting the OR gate perceptron:")
    for inputs, label in zip(X, y):
        prediction = perceptron.predict(inputs)
        print(f"Inputs: {inputs}, Target: {label}, Prediction: {prediction}")

# Train and test the OR gate perceptron
print("Training the OR gate perceptron:")
or_perceptron = Perceptron(input_size=2)
or_perceptron.train(X, y)
test_perceptron(or_perceptron, X, y)

# Visualize the decision boundary
def plot_decision_boundary(perceptron, X, y):
    plt.figure(figsize=(10, 7))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

    x1 = np.linspace(-0.5, 1.5, 10)
    x2 = -(perceptron.weights[0] * x1 + perceptron.bias) / perceptron.weights[1

    plt.plot(x1, x2, c='k', lw=2)
    plt.xlim([-0.5, 1.5])
    plt.ylim([-0.5, 1.5])
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.title('Perceptron Decision Boundary for OR Gate')
    plt.show()

plot_decision_boundary(or_perceptron, X, y)
```

```python
# Print final weights and bias
print(f"\nFinal weights: {or_perceptron.weights}")
print(f"Final bias: {or_perceptron.bias}")
```

```
Training the OR gate perceptron:

Testing the OR gate perceptron:
Inputs: [0 0], Target: 0, Prediction: 0
Inputs: [0 1], Target: 1, Prediction: 1
Inputs: [1 0], Target: 1, Prediction: 1
Inputs: [1 1], Target: 1, Prediction: 1
```




```
Final weights: [0.1 0.1]
Final bias: -0.1
```

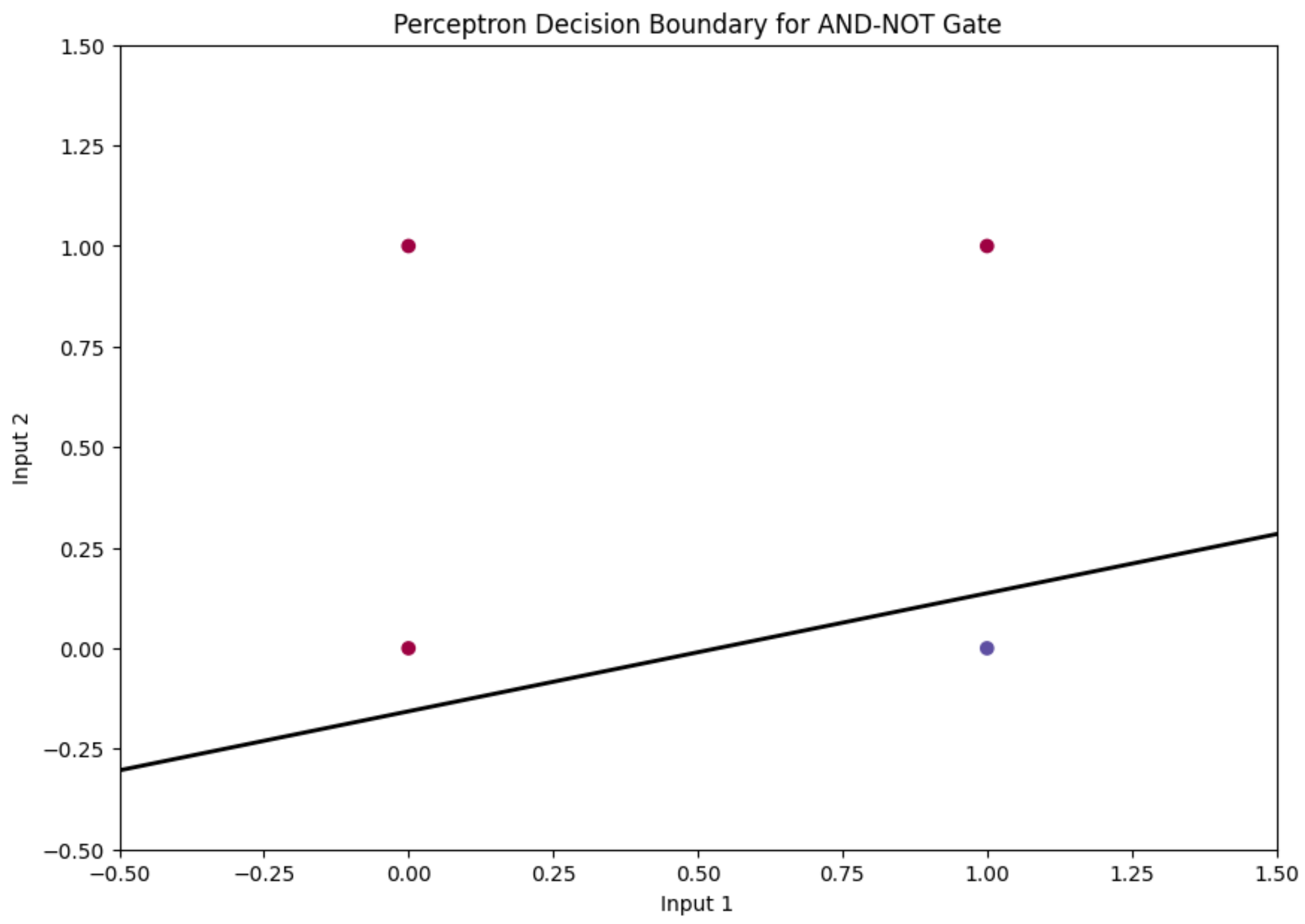
AND-NOT Gate Classification

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

#Create the truth table for the AND-NOT gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 1, 0])

class Perceptron:
```

```
    def __init__(self, input_size, learning_rate=0.1, epochs=1000):
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = np.random.randn(input_size)
        self.bias = np.random.randn()

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        sum = np.dot(inputs, self.weights) + self.bias
        return self.activation_function(sum)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.weights += self.lr * (label - prediction) * inputs
                self.bias += self.lr * (label - prediction)

def test_perceptron(perceptron, X, y):
    print("\nTesting the AND-NOT gate perceptron:")
    correct = 0
    for inputs, label in zip(X, y):
        prediction = perceptron.predict(inputs)
        print(f"Inputs: {inputs}, Target: {label}, Prediction: {prediction}")
        if prediction == label:
            correct += 1
    accuracy = correct / len(y) * 100
    print(f"\nClassification Accuracy: {accuracy}%")

# Train and test the AND-NOT gate perceptron
print("Training the AND-NOT gate perceptron:")
and_not_perceptron = Perceptron(input_size=2)
and_not_perceptron.train(X, y)
test_perceptron(and_not_perceptron, X, y)

# Visualize the decision boundary
def plot_decision_boundary(perceptron, X, y):
    plt.figure(figsize=(10, 7))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

    x1 = np.linspace(-0.5, 1.5, 10)
    x2 = -(perceptron.weights[0] * x1 + perceptron.bias) / perceptron.weights[1

    plt.plot(x1, x2, c='k', lw=2)
    plt.xlim([-0.5, 1.5])
    plt.ylim([-0.5, 1.5])
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.title('Perceptron Decision Boundary for AND-NOT Gate')
```

```
    plt.show()

plot_decision_boundary(and_not_perceptron, X, y)

# Print final weights and bias
print(f"\nFinal weights: {and_not_perceptron.weights}")
print(f"Final bias: {and_not_perceptron.bias}")
```

```
Training the AND-NOT gate perceptron:

Testing the AND-NOT gate perceptron:
Inputs: [0 0], Target: 0, Prediction: 0
Inputs: [0 1], Target: 0, Prediction: 0
Inputs: [1 0], Target: 1, Prediction: 1
Inputs: [1 1], Target: 0, Prediction: 0

Classification Accuracy: 100.0%
```




```
Final weights: [ 0.15127867 -0.51521914]
Final bias: -0.08077389322295916
```

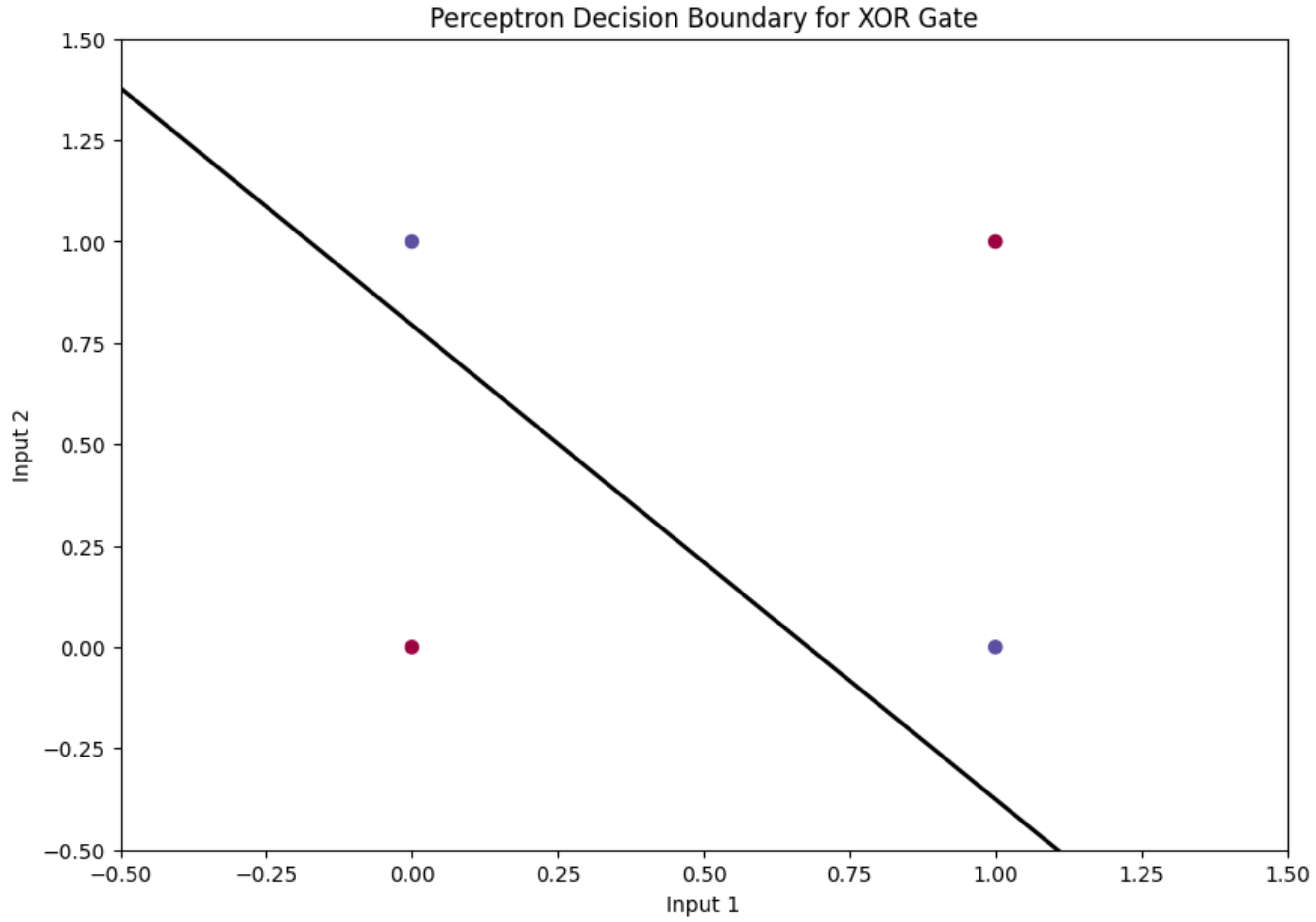
XOR Gate Classification

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

```
# Create the XOR gate's truth table dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=1000):
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = np.random.randn(input_size)
        self.bias = np.random.randn()

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        sum = np.dot(inputs, self.weights) + self.bias
        return self.activation_function(sum)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.weights += self.lr * (label - prediction) * inputs
                self.bias += self.lr * (label - prediction)

def test_perceptron(perceptron, X, y):
    print("\nTesting the XOR gate perceptron:")
    correct = 0
    for inputs, label in zip(X, y):
        prediction = perceptron.predict(inputs)
        print(f"Inputs: {inputs}, Target: {label}, Prediction: {prediction}")
        if prediction == label:
            correct += 1
    accuracy = correct / len(y) * 100
    print(f"\nClassification Accuracy: {accuracy}%")

# Train and test the XOR gate perceptron
print("Training the XOR gate perceptron:")
xor_perceptron = Perceptron(input_size=2)
xor_perceptron.train(X, y)
test_perceptron(xor_perceptron, X, y)

# Visualize the decision boundary
def plot_decision_boundary(perceptron, X, y):
    plt.figure(figsize=(10, 7))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

    x1 = np.linspace(-0.5, 1.5, 100)
    x2 = -(perceptron.weights[0] * x1 + perceptron.bias) / perceptron.weights[1

    plt.plot(x1, x2, c='k', lw=2)
```

```
    plt.xlim([-0.5, 1.5])
    plt.ylim([-0.5, 1.5])
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.title('Perceptron Decision Boundary for XOR Gate')
    plt.show()

plot_decision_boundary(xor_perceptron, X, y)

# Print final weights and bias
print(f"\nFinal weights: {xor_perceptron.weights}")
print(f"Final bias: {xor_perceptron.bias}")
```

```
Training the XOR gate perceptron:

Testing the XOR gate perceptron:
Inputs: [0 0], Target: 0, Prediction: 1
Inputs: [0 1], Target: 1, Prediction: 0
Inputs: [1 0], Target: 1, Prediction: 0
Inputs: [1 1], Target: 0, Prediction: 0

Classification Accuracy: 25.0%
```




```
Final weights: [-0.20954856 -0.17917283]
Final bias: 0.1422855380712728
```