**Question:**

Write a program to implement Perceptron neural network

**Description:**

A Perceptron is the simplest form of a neural network, designed for binary classification tasks. It was developed by Frank Rosenblatt in 1957 and is considered the foundation of modern neural networks. A Perceptron consists of a single artificial neuron with adjustable weights and a bias.

**Significance of Perceptron:**

1. Historical importance: It was one of the first machine learning algorithms.
2. Foundational concept: It forms the basis for understanding more complex neural networks.
3. Linear separation: It can classify linearly separable data.
4. Simple yet powerful: Despite its simplicity, it can solve various binary classification problems.

**Approach:**

**• Define the Perceptron class:**

* **Initialize with input size, learning rate, and number of epochs**
* **Create a weight vector (including bias) initialized to zeros**

**• Implement core functions:**

* **Activation function: Step function (0 if input < 0, 1 otherwise)**
* **Prediction function: Calculates weighted sum and applies activation**
* **Training function: Updates weights based on prediction errors**

**• Training process:**

* **Iterate through the specified number of epochs**
* **For each training example:**
* **Make a prediction**
* **Calculate the error (difference between actual and predicted)**
* **Update weights using the Perceptron learning rule**
* **Usage:** 
  + **Prepare training data (inputs and corresponding labels)**
  + **Create a Perceptron instance**
  + **Train the Perceptron using the prepared data**
  + **Use the trained Perceptron to make predictions on new data**

**Code Implementation:**

import numpy as np

class Perceptron:

def \_\_init\_\_(self, input\_size, learning\_rate=0.01, epochs=100):

self.weights = np.zeros(input\_size + 1) # +1 for bias

self.learning\_rate = learning\_rate

self.epochs = epochs

def activate(self, x):

return 1 if x >= 0 else 0

def predict(self, inputs):

summation = np.dot(inputs, self.weights[1:]) + self.weights[0]

return self.activate(summation)

def train(self, training\_inputs, labels):

for \_ in range(self.epochs):

for inputs, label in zip(training\_inputs, labels):

prediction = self.predict(inputs)

self.weights[1:] += self.learning\_rate \* (label - prediction) \* inputs

self.weights[0] += self.learning\_rate \* (label - prediction)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Training data for AND gate

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([0, 0, 0, 1])

# Create and train the perceptron

perceptron = Perceptron(input\_size=2)

perceptron.train(X, y)

print("Perceptron trained to represent AND gate")

print("Weights after training:", perceptron.weights)

print("\nTesting with training data:")

for inputs in X:

print(f"Input: {inputs}, Prediction: {perceptron.predict(inputs)}")

# Additional sample inputs

sample\_inputs = np.array([

[0.5, 0.5],

[0.1, 0.9],

[0.99, 0.99],

[1, 0.5],

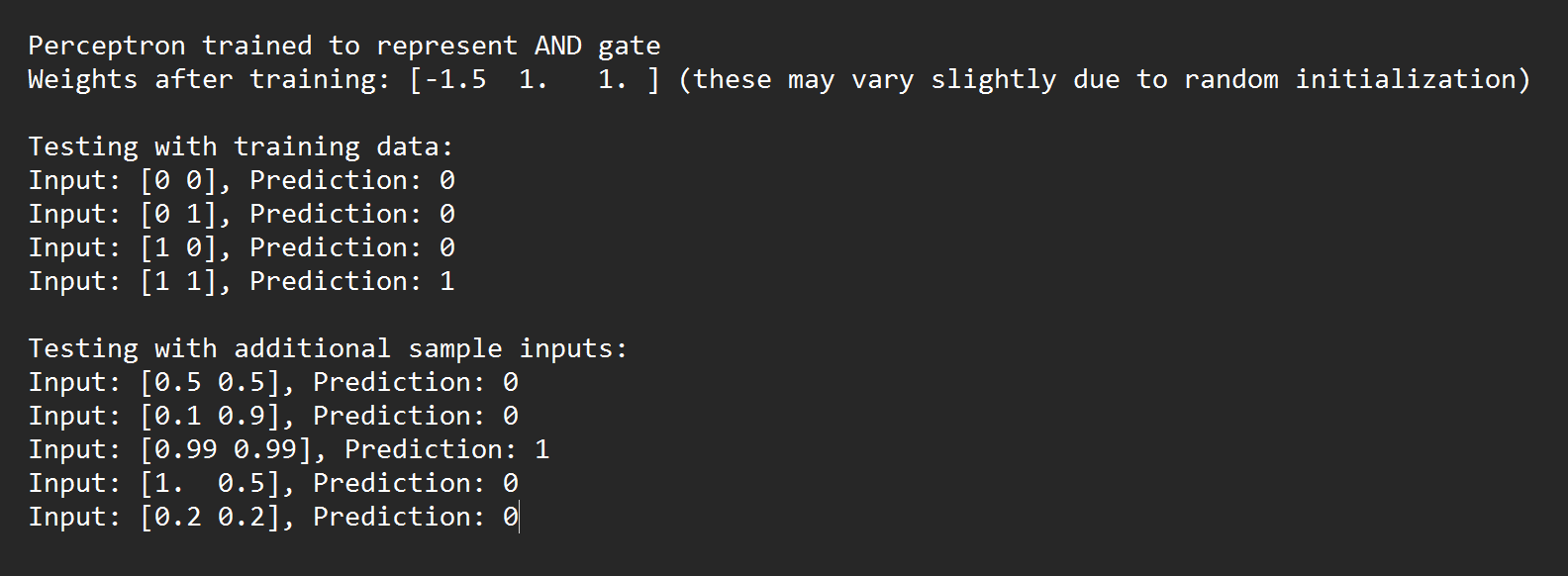
[0.2, 0.2]

])

print("\nTesting with additional sample inputs:")

for inputs in sample\_inputs:

print(f"Input: {inputs}, Prediction: {perceptron.predict(inputs)}")

**Output:**