Perceptron Learning Algorithm

**Objective:**

Write a Program to implement the Perceptron Learning Algorithm using numpy in Python. Evaluate the performance of a single perceptron for NAND and XOR truth tables as input datasets.

**Description:**

A perceptron is a single-layer neural network and the simplest artificial neural network. It consists of input nodes, weights, a bias term, and an activation function.

* Weighted sum is calculated by multiplying the inputs with the weights and adding them.
* Activation function classifies the output using these weighted sums.

If weighted sum >= 0, then output = 1

If weighted sum < 0, then output = 0

**Python Implementation:**

import numpy as np

class Perceptron:

    def \_\_init\_\_(self, input\_size, learning\_rate=0.01, epochs=1000):

        self.weights = np.zeros(input\_size)  # Initialize weights as zeros

        self.bias = 0  # Initialize bias as 0

        self.learning\_rate = learning\_rate

        self.epochs = epochs

    def activation(self, x):

        return 1 if x >= 0 else 0

    def train(self, X, y):

        for \_ in range(self.epochs):

            for inputs, label in zip(X, y):

                # Calculate the weighted sum

                weighted\_sum = np.dot(inputs, self.weights) + self.bias

                # Apply the activation function

                prediction = self.activation(weighted\_sum)

                # Update weights and bias if there's an error

                error = label - prediction

                self.weights += self.learning\_rate \* error \* inputs

                self.bias += self.learning\_rate \* error

    # Predict output for new inputs

    def predict(self, X):

        predictions = []

        for inputs in X:

            weighted\_sum = np.dot(inputs, self.weights) + self.bias

            predictions.append(self.activation(weighted\_sum))

        return predictions

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

    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)

    # Test the perceptron

    test\_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

    predictions = perceptron.predict(test\_input)

    print("Predictions:")

    for i, prediction in enumerate(predictions):

        print(f"Input: {test\_input[i]} => Prediction: {prediction}")

**Description of code:**

1. **Class perceptron –**

* Initializes the weights and bias to 0.
* It contains an activation function that classifies the output.
* A predict function that predicts the output using the activation function.
* A train function that trains the model to find the required weights.
* An evaluate function that calculates the accuracy.

1. **Training and Evaluation for NAND Gate -**

* The NAND truth table (nand\_X) and labels (nand\_y) are defined.
* The perceptron is trained using the train() method.
* The model accuracy is calculated using the evaluate() function.
* The predictions for all inputs in the NAND table are displayed.

1. **Training and Evaluation for XOR Gate**

* The XOR truth table (xor\_X) and labels (xor\_y) are defined.
* The perceptron is trained and evaluated on the XOR data.
* Accuracy is computed, and predictions for all XOR inputs are printed.

**Output:**

Predictions:

Input: [0 0] => Prediction: 0

Input: [0 1] => Prediction: 0

Input: [1 0] => Prediction: 0

Input: [1 1] => Prediction: 1