EX. NO: 01

PERCEPTRON LEARNING

DATE :

AIM:

To design and implement a **Perceptron model** in Python using NumPy that learns the **OR logic gate** through supervised learning. The model should be trained using a step activation function and updated using the **Perceptron Learning Rule**.

ALGORITHM:

Step 1: Import the NumPy library for numerical operations.
Step 2: Define the **step function** as the activation function (returns 1 if input ≥ 0, else 0).
Step 3: Create the **Perceptron class** with:

- Weight initialization (including bias),
- predict() method using weighted sum and activation,
- train() method using the Perceptron learning rule.

Step 4: Prepare **training data** for the OR logic gate (X and y).
Step 5: Instantiate the **Perceptron** object and train it with the data over multiple epochs.
Step 6: After training, test the perceptron by predicting outputs for all possible inputs.
Step 7: Display the final predictions for each input.

PROGRAM:

```
import numpy as np

# Step 1: Define the activation function
# This is a step function: returns 1 if input >= 0, else returns 0
def step_function(value):
    return 1 if value >= 0 else 0

# Step 2: Create the Perceptron class
class Perceptron:
    def __init__(self, input_size, learning_rate=0.1):
        # Initialize weights (including one for bias)
        self.weights = np.zeros(input_size + 1)  # weights = [0.0, 0.0, 0.0]
        self.learning_rate = learning_rate

    # Method to make predictions
    def predict(self, inputs):
        # Add 1 at the beginning of input to represent bias input
        inputs_with_bias = np.insert(inputs, 0, 1)  # e.g., [0, 1] -> [1, 0, 1]
        # Calculate the weighted sum
        total = np.dot(self.weights, inputs_with_bias)
        # Apply step function to decide output
        return step_function(total)

    # Method to train the perceptron
    def train(self, X, y, epochs=10):
        for epoch in range(epochs):
            print(f"Epoch {epoch + 1}")
            for i in range(len(X)):
                prediction = self.predict(X[i])           # Predict output
```

```
            error = y[i] - prediction                    # Calculate error
            x_with_bias = np.insert(X[i], 0, 1)          # Add bias to input
            # Update weights using Perceptron learning rule
            self.weights += self.learning_rate * error * x_with_bias
            print(f" Input: {X[i]}, Predicted: {prediction}, Actual: {y[i]}, Updated Weights:
{self.weights}")

# Step 3: Example usage
if __name__ == "__main__":
    # Training data for OR logic gate
    X = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])

    y = np.array([0, 1, 1, 1])  # Correct output of OR gate

    # Step 4: Create Perceptron and train it
    perceptron = Perceptron(input_size=2)
    perceptron.train(X, y, epochs=10)

    # Step 5: Test the trained Perceptron
    print("\nFinal Predictions:")
    for x in X:
        output = perceptron.predict(x)
        print(f"Input: {x}, Predicted Output: {output}")
```

OUTPUT:

Epoch 1

Input: [0 0], Predicted: 1, Actual: 0, Updated Weights: [-0.1 0.   0.]

Input: [0 1], Predicted: 0, Actual: 1, Updated Weights: [0. 0. 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [0. 0. 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [0. 0. 0.1]

Epoch 2

Input: [0 0], Predicted: 1, Actual: 0, Updated Weights: [-0.1 0.   0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.   0.1]

Input: [1 0], Predicted: 0, Actual: 1, Updated Weights: [0. 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [0. 0.1 0.1]

Epoch 3

Input: [0 0], Predicted: 1, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 4

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 5

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 6

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 7

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 8

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 9

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Epoch 10

Input: [0 0], Predicted: 0, Actual: 0, Updated Weights: [-0.1 0.1 0.1]

Input: [0 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 0], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Input: [1 1], Predicted: 1, Actual: 1, Updated Weights: [-0.1 0.1 0.1]

Final Predictions:

Input: [0 0], Predicted Output: 0

Input: [0 1], Predicted Output: 1

Input: [1 0], Predicted Output: 1

Input: [1 1], Predicted Output: 1

COE (20)	
RECORD (20)	
VIVA (10)	
TOTAL (50)	

RESULT:

The Perceptron model was successfully trained using the OR gate truth table. After several epochs, the weights were adjusted to correctly classify all input combinations. The trained model accurately predicted the OR logic gate outputs.