EX NO: 01

DATE:

IMPLEMENT SIMPLE PERCEPTRON LEARNING

AIM:

To design and implement a single-layer perceptron in Python and train it using the perceptron learning rule to realize the OR logic gate.

ALGORITHM:

Step 1: Start the program.

Step 2: Set all weights to 0, including one extra weight for the bias.

Step 3: Choose a small learning rate (like 0.1) and set the number of training times (epochs).

Step 4: For each input, add a bias value 1 at the beginning.

Step 5: Multiply each input by its weight and add them to get the total (this is the weighted sum)

Step 6: If the total is greater than or equal to 0, the output is 1. Otherwise, the output is 0 (this is the step function).

Step 7: Subtract the predicted output from the actual output to get the error.

Step 8: Update each weight using this formula:
new weight = old weight + (learning rate × error × input)

Step 9: Repeat Steps 4 to 8 for all inputs and for all epochs.

Step 10: After training is complete, test the model with the inputs.

Step 11: Show the final predicted outputs.

Step 12: End the program.

PROGRAM:

```
import numpy as np

# Step 1: Define the activation function
def step_function(value):
    return 1 if value >= 0 else 0
```

```python
# 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)  # bias weight included
        self.learning_rate = learning_rate

    # Method to make predictions
    def predict(self, inputs):
        # Add bias input at the beginning
        inputs_with_bias = np.insert(inputs, 0, 1)
        total = np.dot(self.weights, inputs_with_bias)
        return step_function(total)

    # Method to train the perceptron
    def train(self, X, y, epochs=10):
        for epoch in range(epochs):
            print(f"\nEpoch {epoch + 1}")
            for i in range(len(X)):
                prediction = self.predict(X[i])
                error = y[i] - prediction
                x_with_bias = np.insert(X[i], 0, 1)
                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])  # Expected output for OR gate

    # Step 4: Create and train the perceptron
    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 successfully learned the OR gate and correctly predicted the output for all input combinations after training.