EX NO: 01

DATE:

IMPLEMENT SIMPLE PERCEPTRON LEARNING

AIM:

To implement a single-layer perceptron in Python that can learn and predict the output of an OR logic gate using supervised learning.

ALGORITHM:

Step 1: Start the program.

Step 2: Import required libraries (e.g., NumPy).

Step 3: Define the step activation function: return 1 if input ≥ 0, else return 0.

Step 4: Create the Perceptron class with __init__, predict, and train methods.

Step 5: In the constructor (__init__), initialize all weights (including bias) to zero.

Step 6: Set the learning rate to a small positive value (e.g., 0.1).

Step 7: In the predict method, add a bias input (value 1) to the input vector.

Step 8: Calculate the weighted sum using dot product of weights and input vector.

Step 9: Apply the step activation function to the weighted sum to get prediction.

Step 10: In the train method, repeat training for a fixed number of epochs.

Step 11: For each training input, predict the output using the predict method.

Step 12: Calculate the error: error = actual output − predicted output.

Step 13: Update each weight using the rule:

weights = weights + learning_rate × error × input_with_bias.

Step 14: After training is complete, test the model and print final predictions.

Step 15: End the program.

CODING:

```
import numpy as np

# Step 1: Define the activation function
# This is a step function: returns 1 if input is >= 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)  # example: inputs =
[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"\nEpoch {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 input
                # Update weights using 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 successfully learned the OR gate and correctly predicted the output for all input combinations after training.