Ex.No:

Date:

IMPLEMENT A SIMPLE PERCEPTRON LEARNING

AIM:

To design and implement a simple Perceptron learning algorithm in Python using NumPy that learns the behavior of the OR logic gate through supervised training and correctly classifies input patterns after training.

STEPS:

- First, we import the NumPy library, which helps us work with numbers and arrays easily.
- We define a small function called the step function that checks if a number is greater than or equal to 0. If it is, it gives 1; otherwise, it gives 0.
- We then create a Perceptron class. In the beginning, it sets all weights to 0, and it adds an extra one for the bias. We also set a learning rate.
- Inside the Perceptron class, we make a function called predict. This adds a bias input of 1 to the beginning, then calculates the result using the weights and input values, and finally applies the step function.
- We create another function called train that will run several times (epochs). It goes through each example in the training data to help the model learn.
- During training, the model compares its predicted output to the correct one. If it's wrong, it calculates the error and adjusts the weights to improve.
- We create the input data for the OR gate. These are all the possible input combinations like [0,0], [0,1], [1,0], and [1,1], along with their correct outputs.
- We make a Perceptron object and tell it that we have 2 inputs.
- We train our Perceptron by giving it the input and output data and let it run for 10 rounds (epochs) to learn the OR gate properly.
- Finally, we test the model to see if it gives the correct output for each input. If everything went well, it should correctly show how the OR gate works.

CODE:

```
# Step 1: Import necessary libraries
import numpy as np

# Step 2: Define the step activation function
def step_function(value):
    return 1 if value >= 0 else 0

# Step 3: Define the Perceptron class
class Perceptron:
    def __init__(self, input_size, learning_rate=0.1):
        # Initialize weights (including bias)
        self.weights = np.zeros(input_size + 1)  # +1 for bias
        self.learning_rate = learning_rate

    # Step 4: Define the prediction method
    def predict(self, inputs):
        inputs_with_bias = np.insert(inputs, 0, 1)  # Insert bias input = 1
        total = np.dot(self.weights, inputs_with_bias)  # Weighted sum
        return step_function(total)  # Activation output

    # Step 5: Define the training method
    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)  # Add bias to inputs
            self.weights += self.learning_rate * error * x_with_bias
            print(f" Input: {X[i]}, Predicted: {prediction}, Actual: {y[i]}, Updated Weights: {self.weights}")

# Step 6: Define training data for OR gate
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

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

# Step 7: Create a Perceptron instance
perceptron = Perceptron(input_size=2)

# Step 8: Train the perceptron
perceptron.train(X, y, epochs=10)

# Step 9: Test predictions after training
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
```

RESULT:

Thus, the Perceptron model has successfully learned the OR logic gate.