Exp No: 4
Date: 20/9/25

A Python Program to implement single Layer Perception

Aim:-

To implement python program for the single layer perceptron

Algorithm :-

Step 1:- Import Necessary Libraries

* Define the number of input feactures (input_dim).
* Initialize weights (w) and bias (b) to 0 or small random values.

Step 2:- Initiatize the perceptron

* Define the number of input feauture (input_dim)
* Initialize weights (w) and bias (b) to zero or small random values.

Step 3:- Define Activation function!

* Choose an activation function (eg. step function sigmoid, or ReLU)
* User Defined function - sigmoid_func(x)
  → Compute 1/(1+np.exp(-x)) and return the value
* User defined function - der(x)
  → Compute the product of value of sigmoid_func(x) and (1-sigmoid_func(x)) and return the value

Step 4: Define Training Data:

* Define input features (x) and corresponding target labels (y)

Step 5: Define Learning Rate and Number of Epochs:

* Choose a learning rate (alpha) and the number of training epochs

Step 6: Training the perceptron:

* For each epoch:
  - For each input sample in the training data.
    → Compute the weighted sum of inputs (z) as the dot product of input features and weights plus bias (z = np.dot(x[i], w) + b).
    → Apply the activation function to get the predicted output (y_pred)
    → Compute the error (error = y[i] - y_pred)
    → Update the weights and bias using the learning rate and error (w += alpha * error * x[i]; b += alpha * error)

Step 7: Prediction

* Use the trained perceptron to predict the output for new input data.

Step 8: Evaluate the model

* Measure the performance of the model using metrics such as accuracy, precision, recall etc.

Program:-

```
import numpy as np
import pandas as pd
# input and output data
Input_value = np.array([[0,1],[0,1],[1,1],[1,0]]) #
shape (4,2)
Output = np.array([0,0,1,0]).reshape(4,1) #
shape (4,1)
# intialize weights and bias
weights = np.array([[0,1],[0,3]]) # shape(2,1)
bias = 0.2
# Sigmoid activation function
def sigmoid_func(x)
  return 1/(1+np.exp(-x))
# Derivation of sigmoid
def der(x):
  return sigmoid_func(x) * (1-sigmoid_func
                                      (x))
# Training loop
for epoch in range(15000):
  input_arr = input_value
# Forward pass.
  weighted_sum = np.dot(input_arr, weights)
  + bias
  first_output = sigmoid_func(weighted_sum)
# Error
error = first_output - output
total_error = np.square(error).mean()
```

```
# Backpropogation
first_der = error
second_der = der(weighted_sum) # Derivation
should be on the input to sigmoid}
derivative = first_der * second_der
# Update weights
t_input = input_arr.T
final_derivative = np.dot(t_input, derivative)
weights = weights -(0.05 * final_derivative)
# Update bias
for i in derivative:
    bias = bias -(0.05*i)

# final weights and bias
print("weights :\n", weights)
print(" Bias :\n", bias)
# Test predictions
def predict(x):
    result = np.dot(x, weights) + bias
    return sigmoid_func(result)
# Predict for each input
test_inputs = [np.array([1,0]), np.array([1,1]),
np.array([0,0]), np.array([0,1])]
for pred in test_inputs:
    res = predict(pred)
    print(f"input: {pred}, output: {res}")
```

Result:

Thus the python program to implement single ~~Layer~~ Perceptron has been executed Successfully