EXP 4

A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON

Aim:

To implement a python program for the single layer perceptron.

Algorithm:

Step 1: Import Necessary Libraries:

● Import numpy for numerical operations.

Step 2: Initialize the Perceptron:

● Define the number of input features (input_dim).

● Initialize weights (W) and bias (b) to zero or small random values.

Step 3: Define Activation Function:

● Choose an activation function (e.g., step function, sigmoid, or ReLU).

● User Defined function - sigmoid_func(x):

o Compute 1/(1+np.exp(-x)) and return the value.

● User Defined function - der(x):

o 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:

o For each input sample in the training data:

o 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).

o Apply the activation function to get the predicted output (y_pred).

o Compute the error (error = y[i] - y_pred).

o 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_value=np.array ([[0,0] ,[0,1], [1,1], [1,0]])
input_value.shape
```

```
#(4,2)
output = np.array([0,0,1,0])
output = output.reshape(4,1)
output.shape
#(4,1)
weights=np.array([[0.1],[0.3]])
weights
#array ([[0.1], [0.3]])
bias = 0.2
def sigmoid_func(x):
    return 1/(1+np.exp(-x))
def der(x):
    return sigmoid_func(x)*(1 - sigmoid_func(x))
for epochs in range(15000):
    input_arr = input_value
    weighted_sum=np.dot(input_arr,weights)+bias
    first_output=sigmoid_func(weighted_sum)
    error=first_output - output
    total_error=np.square(np.subtract(first_output,output)).mean()
    first_der=error
    second_der=der(first_output)
    derivative=first_der*second_der
    t_input = input_value.T
    final_derivative=np.dot(t_input,derivative)
    weights=weights - (0.05 * final_derivative)
    for i in derivative:
        bias=bias-(0.05*i)
print(weights)
print(bias)
#[16.57299223]
#[16.57299223]]
#[-25.14783487]
pred=np.array([1,0])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.00018876]
pred=np.array([1,1])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.99966403]
pred=np.array([0,0])
result = np.dot(pred,weights)+bias
```

```
res = sigmoid_func(result)
print(res)
#[1.19793729e-11]
pred=np.array([0,1])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.00063036]
```

OUTPUT:

```
(4, 2)
```

```
(4, 1)
```

```
array([[0.1],
       [0.3]])
```

```
[[6.62916366]
 [6.62916441]]
[-10.23197316]

 [0.02652435]
```

```
[0.95375065]
```

```
[3.59993686e-05]
```

```
[0.02652437]
```

RESULT:

Thus, the Python program to implement a single-layer perceptron has been executed successfully.