Experiment: 1

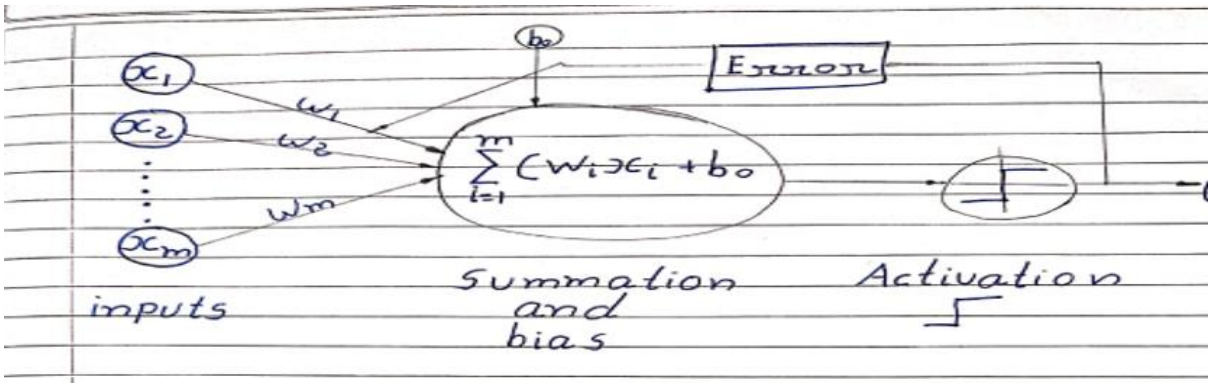
AIM: To study the architecture of single layer perceptron.

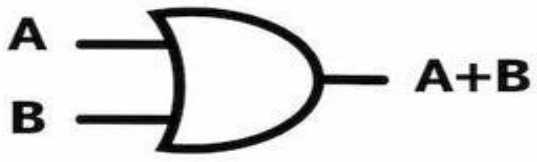
Theory:

- Single Layer Perceptron is one of the oldest and first introduced neural networks. It was proposed by Frank Rosenblatt in 1958. Perceptron is also known as an artificial neural network. Perceptron is mainly used to compute the logical gate like AND, OR, and NOR which has binary input and binary output.

- The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.

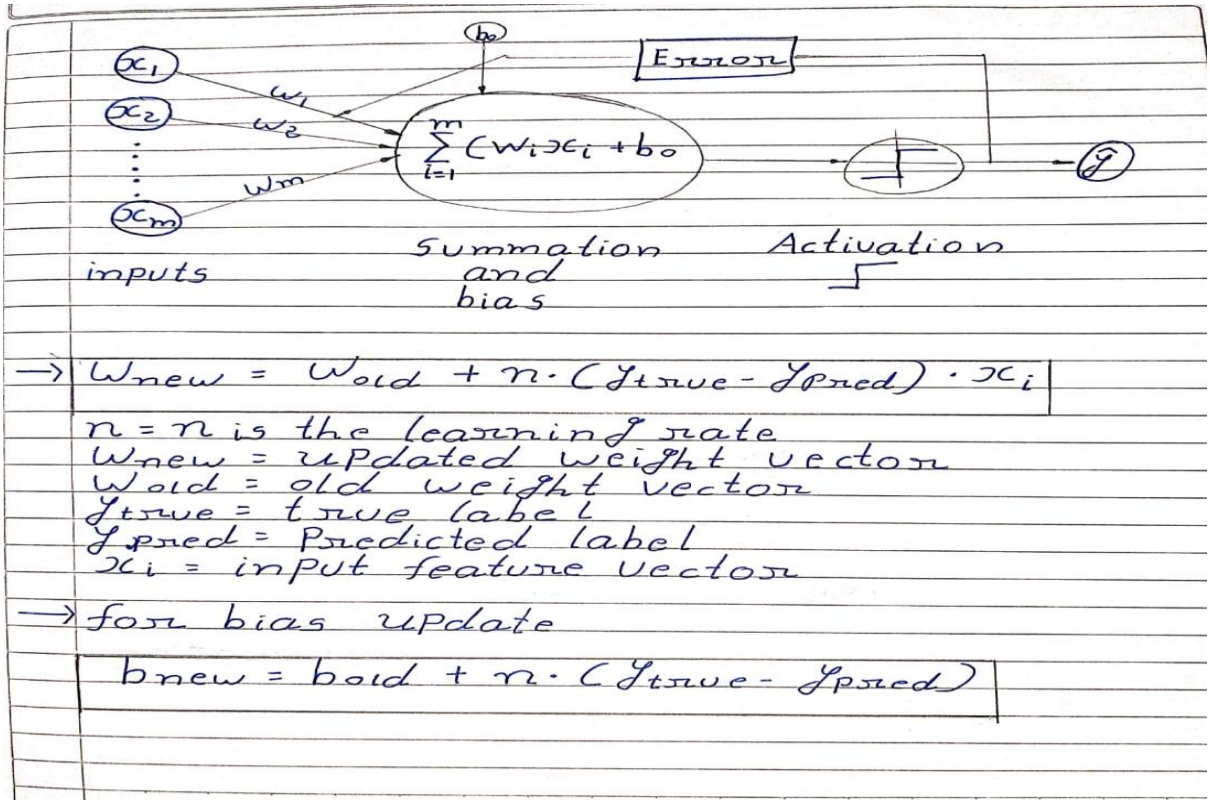
- Methodology : input pattern representing different combination of or gate inputs are defined weights and bias are defined they can also updates perceptron is trained over multiple epochs adjusting error
- This is also visualized with the plotting over the error over epochs
- Intuition :The or gate is classic binary logic function and this program aims to perceptron mimic behaviour of gate





2 input OR gate

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



$$W_{new} = W_{old} + \eta \cdot (y_{true} - y_{pred}) \cdot x_i$$

η = η is the learning rate
W_{new} = updated weight vector
W_{old} = old weight vector
y_{true} = true label
y_{pred} = Predicted label
x_i = input feature vector

→ for bias update

$$b_{new} = b_{old} + \eta \cdot (y_{true} - y_{pred})$$

- The formula for the output (y) of a Single Layer Perceptron (SLP) can be expressed as follows:

$$Y = f(\sum wi * xi + b)$$

Where:

- Y is the output of the perceptron.
- f is an activation function, often a step function or a sigmoid function.
- wi represents the weights associated with the input features.
- xi represents the input features.
- b is the bias term.

- The Single Layer Perceptron (SLP) functions by taking input features from the input layer and associating each with corresponding weights. A weighted sum of the inputs, combined with a bias term, is computed, and the result is passed through an activation function to produce the perceptron's output.

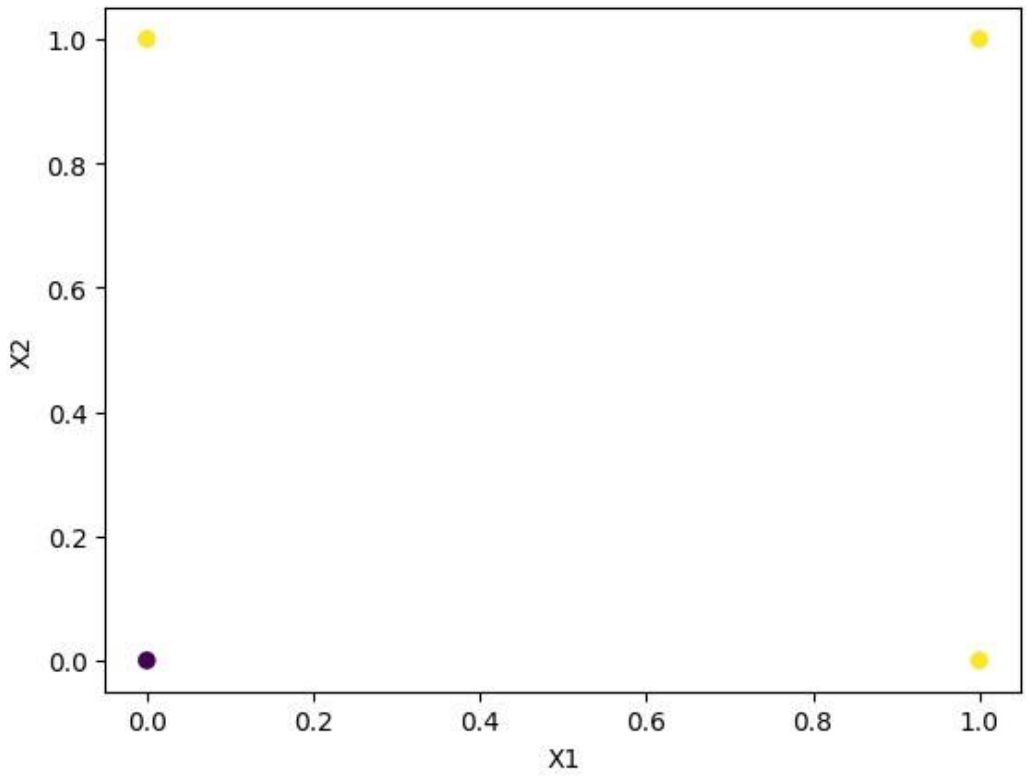
- The activation function introduces non-linearity to enable learning of complex relationships. During training, the weights and bias are adjusted iteratively to minimize the error between predicted and actual outputs, a process that continues until satisfactory performance is achieved or until a specified number of iterations are reached.

Code:

```
Import numpy as np
Import matplotlib.pyplot as plt

x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
w = np.array([[0], [1]], dtype=float)
bias = 0.2
ita = 0.01
y_hat = np.array([[0], [1], [1], [1]])
y_pred = np.array([[0], [0], [0], [0]])
error_history = []
bias_history = []
w1 = []
w2 = []
epochs = 25
for epoch in range(epochs):
    error_sum = 0
    for i in range(len(x)):
        weight_sum = np.dot(x[i], w) + bias
        if weight_sum >= 0:
            y_pred[i] = 1
        else:
            y_pred[i] = 0
        error = y_hat[i] - y_pred[i]
        error_history.append(error)
        w1.append(w[0][0])
        w2.append(w[1][0])
        w += ita * (y_hat[i] - y_pred[i]) * x[i].reshape(-1, 1)
        bias += ita * (y_hat[i] - y_pred[i])
        bias_history.append(bias[0])

print("weighted sum" , w)
print("bias " , bias)
print("predicted value "  , y_pred)
print("actual value" ,   y_hat)
```

Output:

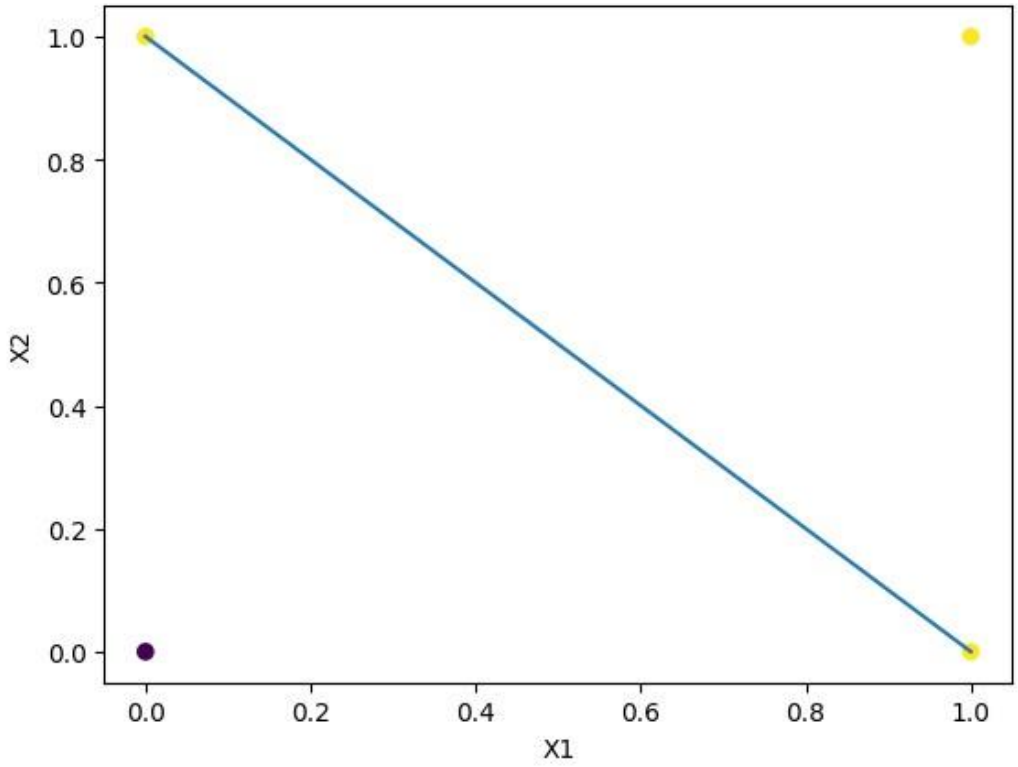
- **Plotting the input data:**



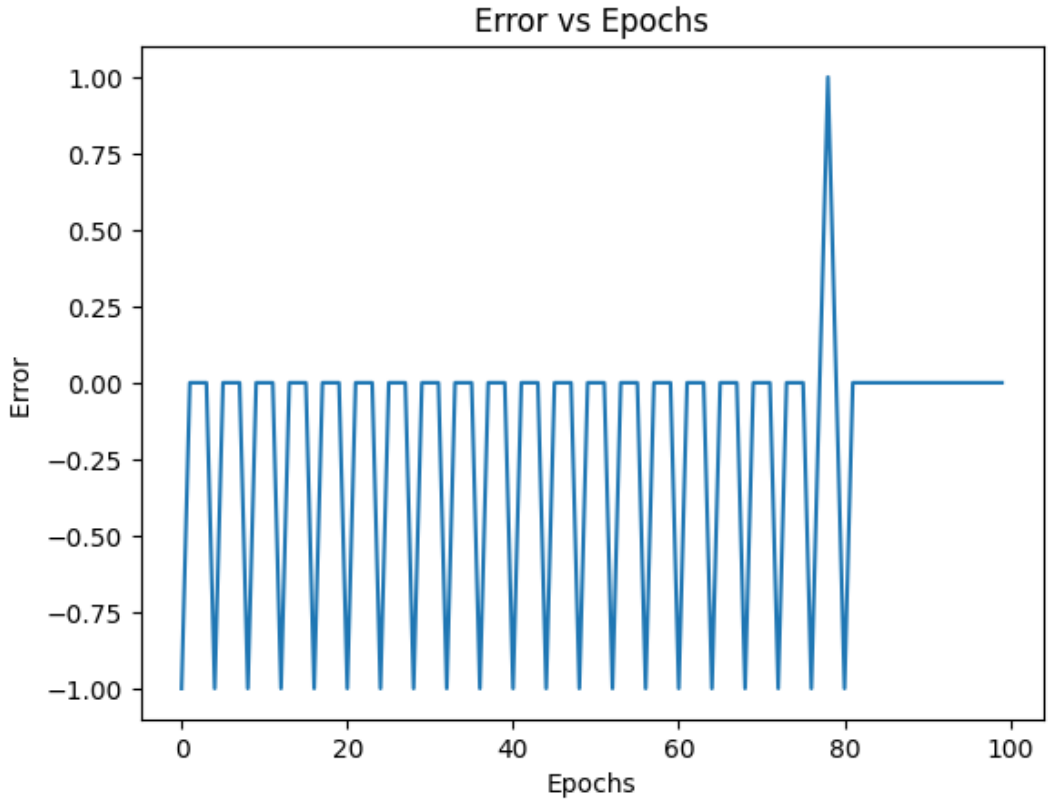
- **Making predictions on the trained model:**

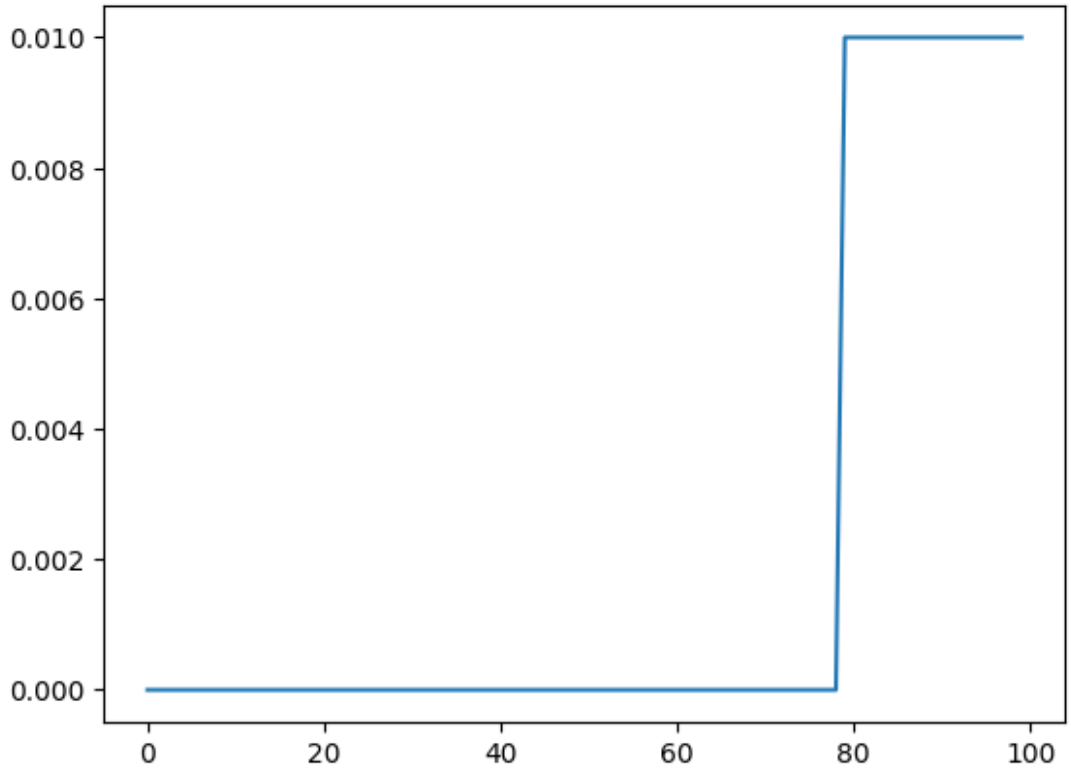
```
Prediction = 0, Expected = 0
Prediction = 1, Expected = 1
Prediction = 1, Expected = 1

Prediction = 1, Expected = 1
```

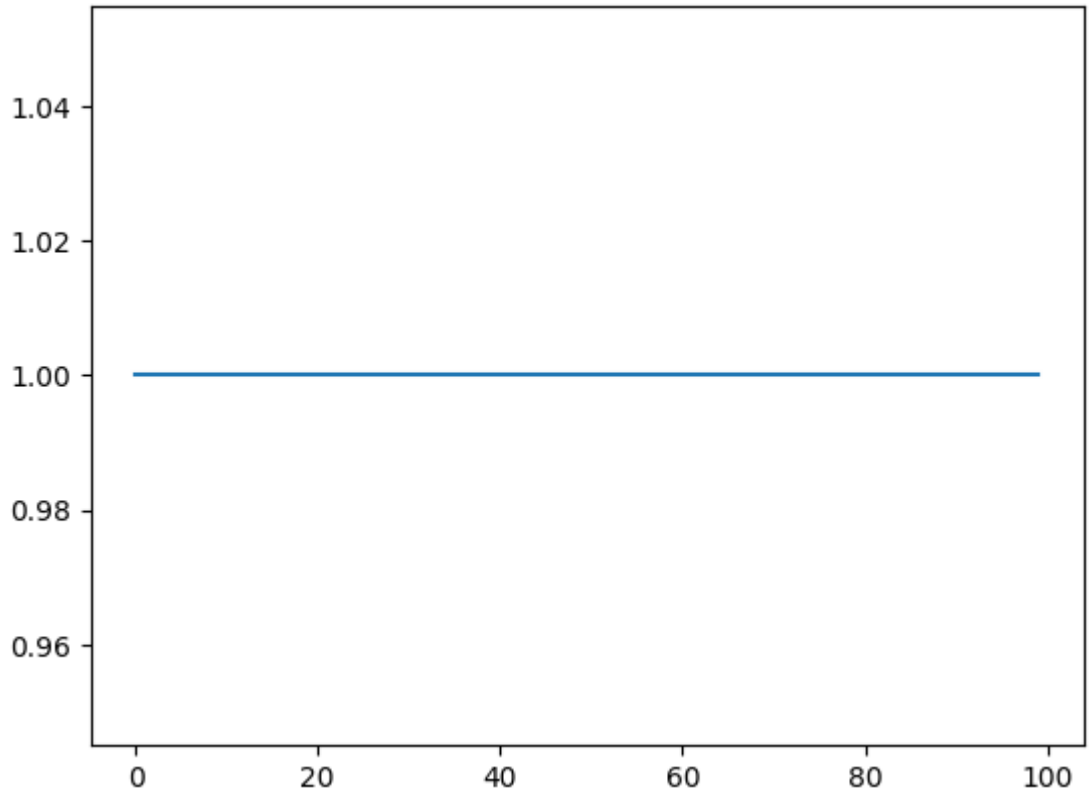
- **Visualizing the decision boundary:**



- **Plotting the changes in bias and weights during training:**




0.010
0.008
0.006
0.004
0.002
0.000
0
20
40
60
80
100


1.04
1.02
1.00
0.98
0.96
0
20
40
60
80
100

- **Displaying the final weights:**

```
Final Weight 1: 0.01
Final Weight 2: 1.00
```

Conclusion:

- ⇨ Along with actual and predicted values are same it also show successful implement of or gate as last we can see that error rate is 0 in epochs .
- ⇨ Conclusion : the perceptron effectively learning or gate logic showing its ability to make binary classification based on learned parameter
- ⇨ predicted value [[0] [1] [1] [1]] actual value [[0] [1] [1] [1]]
- ⇨ Results : the final weight sum and bias is weighted sum [[0.01] [1.]] => bias [-3.12250226e-17]