Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

A.Y. 2022-2023

Subject: Artificial Intelligence **SAP ID: 60004220253 – Devansh Mehta**

Experiment No. 06

Aim: Program to implement learning : Perceptron Learning

Theory:

Perceptron is one of the simplest Artificial neural network architectures. It was introduced by Frank Rosenblatt in 1957s. It is the simplest type of feedforward neural network, consisting of a single layer of input nodes that are fully connected to a layer of output nodes. It can learn the linearly separable patterns. it uses slightly different types of artificial neurons known as threshold logic units (TLU). it was first introduced by McCulloch and Walter Pitts in the 1940s.

Basic Components of Perceptron

A perceptron, the basic unit of a neural network, comprises essential components that collaborate in information processing.

Input Features: The perceptron takes multiple input features, each input feature represents a characteristic or attribute of the input data.

Weights: Each input feature is associated with a weight, determining the significance of each input feature in influencing the perceptron's output. During training, these weights are adjusted to learn the optimal values.

Summation Function: The perceptron calculates the weighted sum of its inputs using the summation function. The summation function combines the inputs with their respective weights to produce a weighted sum.

Activation Function: The weighted sum is then passed through an activation function. Perceptron uses Heaviside step function functions. which take the summed values as input and compare with the threshold and provide the output as 0 or 1.

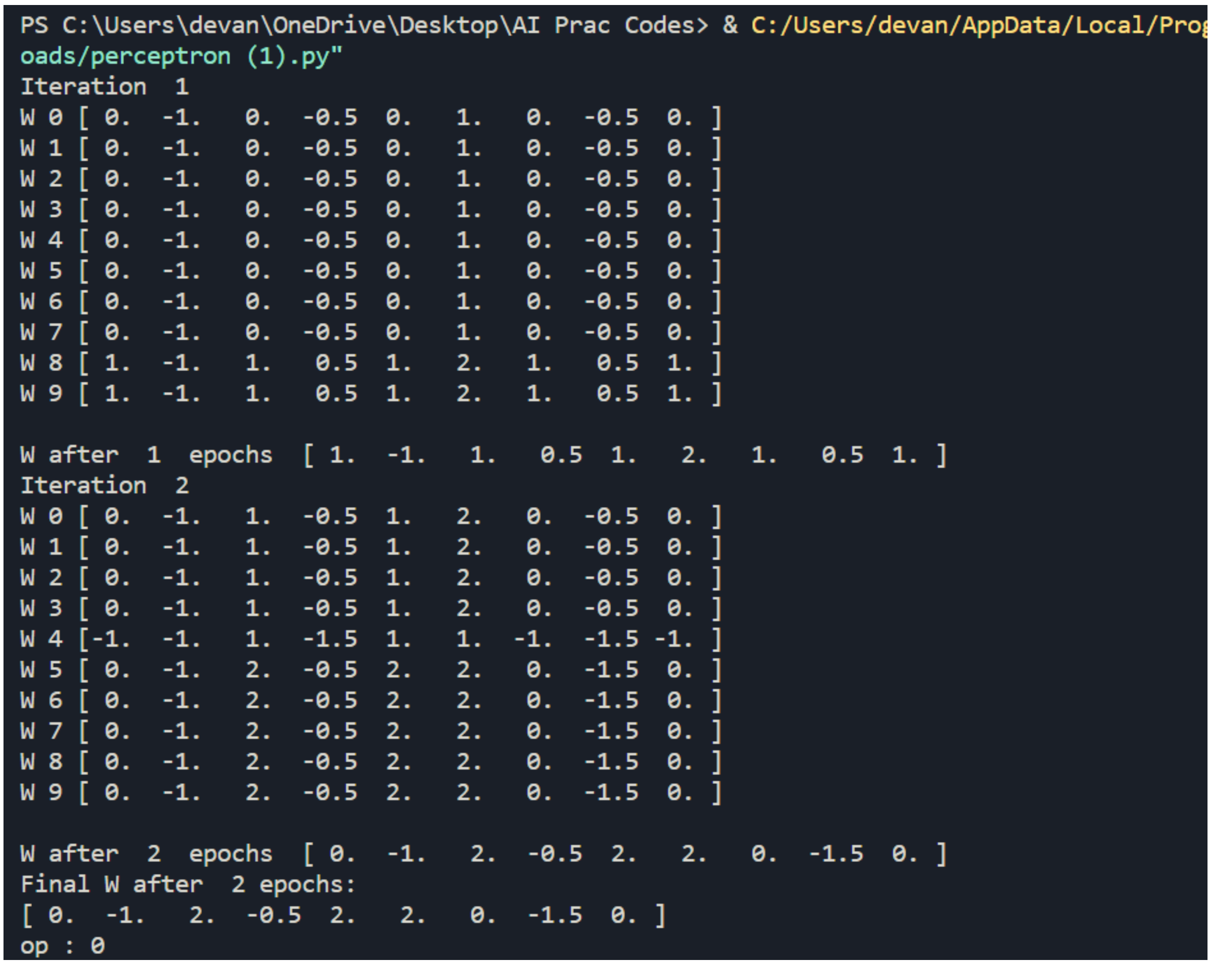
Output: The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).

Bias: A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.

Learning Algorithm (Weight Update Rule): During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron learning algorithm, which updates weights based on the difference between the predicted output and the true output.

Code:

```
import numpy as np

X1 = np.array([1,0,0,1,0,0,1,1,1])
X2 = np.array([1,0,0,1,0,0,1,1,1])
X3 = np.array([1,1,0,1,0,0,1,1,1])
X4 = np.array([1,0,0,1,0,0,1,1,1])
X5 = np.array([1,0,0,1,0,1,1,1,1])
X6 = np.array([1,0,1,1,1,1,1,0,1])
X7 = np.array([1,0,1,1,1,1,1,0,1])
X8 = np.array([1,0,1,1,1,1,1,0,1])
X9 = np.array([1,0,1,1,1,1,1,1,1])
X10 = np.array([1,1,1,1,1,1,1,0,1])

X = np.array([X1, X2, X3, X4, X5,X6,X7,X8,X9,X10])
W = np.array([1, -1, 0, 0.5,0,1,1,0.5,1])

d = np.array([0,0,0,0,0,1,1,1,1,1])

c = 1
epochs = 2
i=0

for i in range(epochs):
    print("Iteration ", i+1)
    for j in range(len(X)):
        net = np.dot(X[j], W)

        if (net <= 0):
            op = 0
        elif net > 0:
            op = 1

        error = d[j] - op

        dW = c*error*X[j]
        W += dW
        print("W", j,  W)

    print("\nW after ", i+1, " epochs ", W)
    # c=c/2
print("Final W after ", epochs, "epochs:")
print(W)
j=0

net = np.dot([1,0,0,1,0,0,1,1,0], W)

if (net <= 0):
    op = 0
elif net > 0:
    op = 1

print("op",":", op)
```



```
PS C:\Users\devan\OneDrive\Desktop\AI Prac Codes> & C:/Users/devan/AppData/Local/Prog
oads/perceptron (1).py"
Iteration  1
W 0 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 1 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 2 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 3 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 4 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 5 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 6 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 7 [ 0.  -1.   0.  -0.5  0.   1.   0.  -0.5  0. ]
W 8 [ 1.  -1.   1.   0.5  1.   2.   1.   0.5  1. ]
W 9 [ 1.  -1.   1.   0.5  1.   2.   1.   0.5  1. ]

W after  1  epochs  [ 1.  -1.   1.   0.5  1.   2.   1.   0.5  1. ]
Iteration  2
W 0 [ 0.  -1.   1.  -0.5  1.   2.   0.  -0.5  0. ]
W 1 [ 0.  -1.   1.  -0.5  1.   2.   0.  -0.5  0. ]
W 2 [ 0.  -1.   1.  -0.5  1.   2.   0.  -0.5  0. ]
W 3 [ 0.  -1.   1.  -0.5  1.   2.   0.  -0.5  0. ]
W 4 [-1.  -1.   1.  -1.5  1.   1.  -1.  -1.5 -1. ]
W 5 [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
W 6 [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
W 7 [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
W 8 [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
W 9 [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]

W after  2  epochs  [ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
Final W after  2 epochs:
[ 0.  -1.   2.  -0.5  2.   2.   0.  -1.5  0. ]
op : 0
```

Conclusion: Thus, we implemented Perceptron Learning Rule.