EXPERIMENT NO : 8

AIM :

To implement A perception learning Model

THEORY :

Perceptions are one of the earliest neural network model given by Rosenblatt in 1962.

A perception models a neuron by taking a weighted sum of its inputs and sending the output 1 . if the sum is greater than some adjustable threshold value, otherwise it sends 0 .

It is unidirectional connection model.

Step activation function binary input / output as either 0 (or) 1

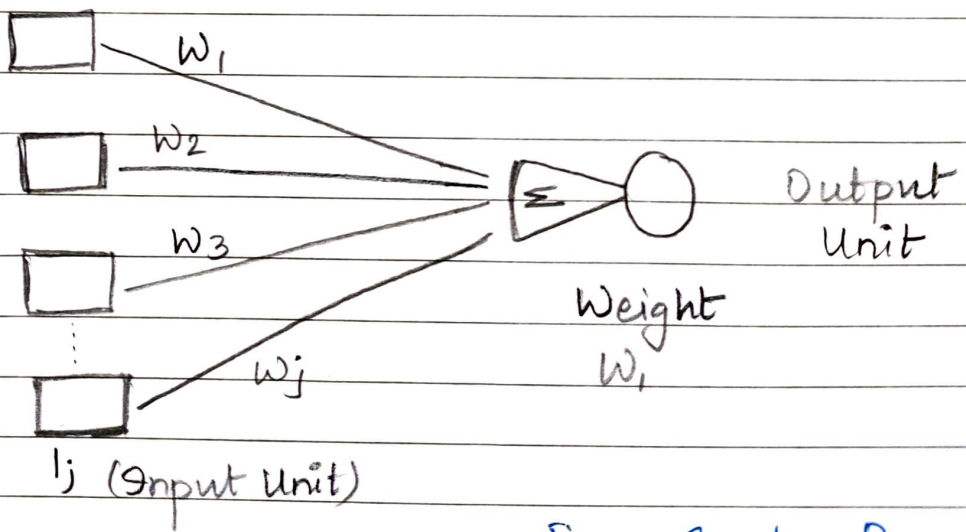

Fig: Single Perception.

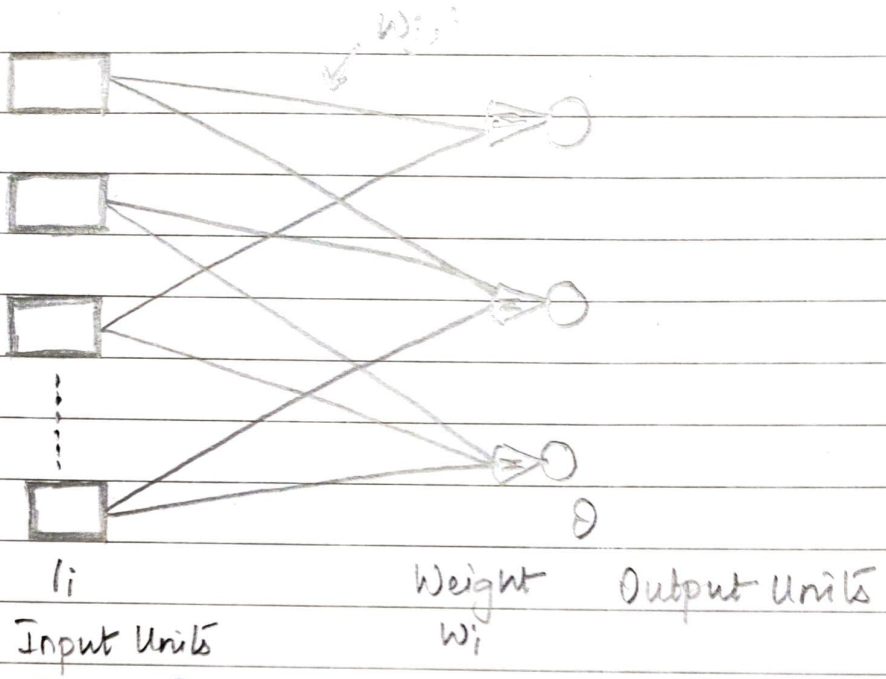

Fig : Perception Network.

I_j - Activation of a unit j in the input layer

W_j - Weight from unit j to the output in a perception.

O - Activation of the single output unit of a perception.

O_i - Activation of unit i in the output layer.

$W_{j,i}$ - Weight of the link from unit j to unit i.

Therefore, the activation of the output unit is :

$$O = \text{Step}_0 \left[\sum_i W_j I_j \right] = \text{Step}_0 (W, I)$$

W - vector of all weights in the network.

I - vector of activation of all input units.

SINGLE LAYER PERCEPTION

Feed - Forward networks are normally arranged in layers. The simplest one having only one layer, no hidden layers is called as perception n/w.

The output units all operate separately and there are no shared weights.

Adjusting weights move the location, orientation and steepness of cliff.

Steps: Calculate Net

If net is greater than 0 +ve $O(i) = 1$

If net is less than 0 -ve $O(i) = -1$

Calculate $\Delta w = C w x$ where $n = (di - Oi)$

$W_2 = W_1 + \Delta W_1$ (for single iteration)

CONCLUSION:

Thus we have successfully implemented the perception learning model. Using Python Programming Language and understood its basic concepts.

Roll No: 5117060

Aim: To implement a program demonstrating the working of a perceptron.

Code:

```
def multiply(weight, x):
    sum = 0
    for i in range(len(weight)):
        sum += weight[i] * x[i]
    return sum
N = int(input('\n Enter number of inputs:'))
c = float(input('\n Enter learning constant: '))
desired_op=[]
input_x = []
for i in range (N):
    temp = list(map(float,input('\n Enter x vector:').split(',')))
    input_x.append(temp)
    t = float(input('\n Enter desired output:'))
    desired_op.append(t)
weight = list(map(float,input('\n Enter weights:').split(',')))
print('\n Input vectors:',input_x)
print('\n Desired outputs:',desired_op)
print('\n Learning rate:',c)
print('\n Weights:',weight)
iterate = int(input('\n Enter number of iterations:'))
for i in range(iterate):
    print('\n Iteration Number:',i+1)
    for j in range(N):
        print('\n Input number:',j+1)
        net = multiply(weight,input_x[j])
        print('\n Net[',j+1,']=',net)
        if (net <= 0):
            o = 0
        else:
```

```
            o = 1.0
        print('\n Actual Output:{0} Desired Output
{1}'.format(o,desired_op[j]))
        if o == desired_op[j]:
            break
        print("\n Since Actual Output is not equal to desired
output.\nTherefore, change Weights")
        delta =list(c*(desired_op[j] - o) * k for k in input_x[j])
        print('\n Delta_w =',delta)
        for m in range(len(weight)):
            weight[m] += delta[m]
        print('\n Updated weights:',weight)
```

Output:

```
Enter number of inputs:3
 Enter learning constant: 1
 Enter x vector:1,2
 Enter desired output:1
 Enter x vector:-1,2
 Enter desired output:0
 Enter x vector:0,-1
 Enter desired output:0
 Enter weights:1.0,-0.8
 Input vectors: [[1.0, 2.0], [-1.0, 2.0], [0.0, -1.0]]
 Desired outputs: [1.0, 0.0, 0.0]
 Learning rate: 1.0
 Weights: [1.0, -0.8]
 Enter number of iterations:1
 Iteration Number: 1
 Input number: 1
 Net[ 1 ]= -0.6000000000000001
```

```
 Actual Output:0 Desired Output 1.0

 Since Actual Output is not equal to desired output.
Therefore, change Weights

 Delta_w = [1.0, 2.0]

 Updated weights: [2.0, 1.2]

 Input number: 2

 Net[ 2 ]= 0.3999999999999999

 Actual Output:1.0 Desired Output 0.0

 Since Actual Output is not equal to desired output.
Therefore, change Weights

 Delta_w = [1.0, -2.0]

 Updated weights: [3.0, -0.8]

 Input number: 3

 Net[ 3 ]= 0.8

 Actual Output:1.0 Desired Output 0.0

 Since Actual Output is not equal to desired output.
Therefore, change Weights

 Delta_w = [-0.0, 1.0]

 Updated weights: [3.0, 0.19999999999999996]
>>>
```