Page No.
Date

Experiment 6

Shashwat Shah
60004220126
Div B C22

Aim: To study and implement perceptron learning algorithm.

Theory: Perception learning is a fundamental concept in machine learning and artificial neural networks. The perceptron was developed by Frank Rosenblt in 1957 and is considered one of the earliest neural network models.

Structure of a perception includes the input nodes associated with weights and an output node. The inputs and weights are multiplied and the weighted sum is passed through an activation function. The output is a binary decision based on whether the weighted sum exceeds a certain threshhold which is usually a step function

The process involves the following steps :-

i) Initialization - Initialize the inputs (x) and weights (w) and the learning constant (c)

ii) Calculation for each input compute the weighted sum and get the output theory

iii) Error calculation and weight update.

$$\Delta w = c \cdot (\text{desired} - \text{output}) x_i$$

$$w_{i+1} = \Delta w + w_i$$

iv) Repeat steps ii & iii for a specific number of iterations until the weight converge.

Page No.
Date

Advantage -

- Efficient method and very easy to understand and implement
- It laid the foundation of more complex neural networks.
- These are yet used in certain applications in context of neural network history & development.

Limitation -

Perception learning algorithm may not converge if the data is not linearly seperable.

It is sensitive to the choice of initial weights and learning rate.

Conclusion: Perception learning is a straight forward method for training a basic neural network model while it has limitations it played a crucial role in history of neural network and set the foundation for sophisticated models.

Experiment – 7

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Perceptron training algorithm for L and M classification

Theory:
Perceptrons are a type of artificial neuron that predates the sigmoid neuron. It appears that they were invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory.

A perceptron can have any number of inputs, and produces a binary output, which is called its activation.

First, we assign each input a weight, loosely meaning the amount of influence the input has over the output.
To determine the perceptron's activation, we take the weighted sum of each of the inputs and then determine if it is above or below a certain threshold, or *bias, *represented by b.

The formula for perceptron neurons can can be expressed like this:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Algorithm:

```
def perceptron(inputs, bias)

weighted_sum = sum {
for each input in inputs
input.value * input.weight
}

if weighted_sum <= bias return 0
if weighted_sum > bias
return 1
end
```

Code:

```
def sgn(net_input):
    if net_input <= 0 :
        return -1
    return 1

def pattern_classifier(n_iterations, input, weight, desired_output, learning_rate):
    for iteration in range(n_iterations):
        print(f'Iteration {iteration+1}')
        output = [] for i,X in
        enumerate(input):
            net_input = 0 for j in
            range(len(X)):
                net_input+=weight[j]*X[j]
            generated_output = sgn(net_input)
            output.append(generated_output) if
            generated_output != desired_output[i]:
                difference = desired_output[i] - generated_output for
                position in range(len(weight)):
                    weight[position] = float("{:.2f}".format(weight[position] +
learning_rate*difference*X[position])) print(f'Generated Output vector for
        Iteration {iteration+1} : {output}') print(f'Weight vector after Iteration
        {iteration+1} : {weight}') print("------"*25) if output == desired_output:
            break
    return output,weight

def main(): input
    = [
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1], #L starts here
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,1,1],
        [1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,1,1,1],
        [0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,0,1,1],
        [0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,1,1,0,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,1,1],
```

```
    [0,1,0,1,0,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1], #M starts here
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,1,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,1,0,1,1,0,0,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,0,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,0,0,0,0,0],
    [1,0,0,0,1,1,1,1,1,1,1,0,1,0,1,1,0,0,0,1],
]
desired_output = [1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
initial_weight = [1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1]
learning_rate = 0.05 n_iterations = 3

classification_output, weight_vector = pattern_classifier(n_iterations, input,
initial_weight, desired_output, learning_rate)

count = 0 for i, output in
enumerate(classification_output):
    if output == desired_output[i]: count+=1

accuracy = (count / len(input))*100

print(f'Accuracy of Classifier : {accuracy} %')

print('Classifying an Unknown Sample of L (Output = 1)')
unknown_sample = [1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0]
print('Unknown Sample : ',unknown_sample) net_input=0
for i in range(len(unknown_sample)):
    net_input+=weight_vector[i]*unknown_sample[i]
predicted_output = sgn(net_input)
print('Predicted Output : ', predicted_output)
print("\n") main()
```

Output:

```
Iteration 1
Generated Output vector for Iteration 1 : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1]
```

```
Weight vector after Iteration 1 : [0.2, 0.6, 0.0, 0.6, 0.2, -0.9, 0.4, 0.6, -0.6, 0.1, 0.1, -0.1, 0.4,
0.9, -0.9, 0.1, 1.0, -0.3, 1.0, 0.1]
-------------------------------------------------------------------------------------------------------------------------
Iteration 2
Generated Output vector for Iteration 2 : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]
Weight vector after Iteration 2 : [0.1, 0.5, 0.0, 0.5, 0.1, -1.0, 0.4, 0.5, -0.6, 0.0, 0.0, -0.1, 0.3,
0.9, -1.0, 0.0, 1.0, -0.3, 1.0, 0.0]
-------------------------------------------------------------------------------------------------------------------------
Iteration 3
Generated Output vector for Iteration 3 : [1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1]
Weight vector after Iteration 3 : [0.1, 0.4, 0.0, 0.4, 0.0, -1.0, 0.4, 0.4, -0.6, -0.1, 0.0, -0.1, 0.2,
0.9, -1.0, 0.0, 1.1, -0.2, 1.1, 0.0]
-------------------------------------------------------------------------------------------------------------------------
Accuracy of Classifier : 90.0 %
Classifying an Unknown Sample of L (Output = 1)
Unknown Sample :  [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0]
Predicted Output : 1
```

Conclusion: Hence we have performed **Perceptron training algorithm for L and M classification**