PRACTICAL 2 - PERCEPTRON ALGORITHM FOR A BINARY CLASSIFICATION PROBLEM

AIM:

Train a model that can correctly classify data points into one of two categories based on their features. Gain a deeper understanding of how the Perceptron Algorithm works and how it can be used to solve binary classification problems.

SOFTWARE:

Anaconda3 2021.11 | Spyder 5.1.5 | Python 3.9.7 – 64-Bit

THEORY (PERCEPTRON LEARNING ALGORITHM):

Step 1 – Initialize the weights and biases to random values.

Step 2 – For each training example, compute the predicted output using the current weights and bias.

Step 3 – Update the weights and bias based on the error between the predicted output and the true output. The update rule is given by,

- $w_i = w_i + \text{learning\_rate} * (y - y_{pred}) * x_i$
- $\text{bias} = \text{bias} + \text{learning\_rate} * (y - y_{pred})$

Step 4 – Repeat steps 2 and 3 until the algorithm converges or a maximum number of iterations is reached,

It's also important to note that the Perceptron Algorithm is linear and works best when the data points can be separated with a linear decision boundary. If the data is not linearly separable, the algorithm may not converge or may converge to a suboptimal solution. In such cases, we may need to use non-linear algorithms like SVMs or neural networks.

The perceptron algorithm is a type of supervised learning algorithm used for binary classification problems. In this problem, we have to implement the perceptron algorithm for the three basic logic gates - AND, OR, and XOR. The perceptron algorithm learns from the given labelled data and tries to find the decision boundary that separates the two classes.

PYTHON CODE:

```
import numpy as np

# Define the input features and labels for the logic gates
and_features = np.array([[0,0],[0,1],[1,0],[1,1]])
or_features = np.array([[0,0],[0,1],[1,0],[1,1]])
xor_features = np.array([[0,0],[0,1],[1,0],[1,1]])
```

```
and_labels = np.array([0,0,0,1])
or_labels = np.array([0,1,1,1])
xor_labels = np.array([0,1,1,0])

# Define the step function that returns 1 if the input is greater than or
equal to 0, else 0
def step_fun(sum):
    return np.where(sum >= 0, 1, 0)

# Initialize the weights and bias for the perceptron algorithm
w = np.random.rand(2)
bias = np.random.rand()

# Define the learning rate and the number of epochs for the algorithm
alpha = 0.1
max_epochs = 1000

print("\nWeight: ", w)
print("Bias: ", bias)
print("Alpha: ", alpha)
print("Epoch: ", max_epochs)

# Implement the perceptron algorithm using vectorized operations (instead of a
nested loop) - Calculate the predicted values for all input features at once
def perceptron(features, labels, operator):
    global w, bias

    for i in range(max_epochs):
        print("\nEpoch Value: ", i+1)

        # sum = features[j][0]*w[0] + features[j][1]*w[1] + bias
        sum = np.dot(features, w) + bias

        predicted = step_fun(sum)
        delta = labels - predicted

        # Update weights and bias using the delta rule
        # w[k] += delta * alpha * features[j][k]
        w += alpha * np.dot(features.T, delta)

        # More Efficient Stopping Criterion
        bias += alpha * np.sum(delta)
        error = np.mean(np.abs(delta))

        for j in range(len(features)):
```

```
            print(features[j][0], "", operator, "", features[j][1], "->
Actual:", labels[j], "; Predicted:", predicted[j], " ( w1:", w[0], "& w2:",
w[1], ")")

        # Terminate the algorithm if all the predicted values are correct
        if error == 0:
            break

# Run the perceptron algorithm for different logic gates
while True:
    operator = str(input("\nEnter Logical Operator (and, or, xor, exit): "))

    if operator == "and":
        perceptron(and_features, and_labels, operator)
    elif operator == "or":
        perceptron(or_features, or_labels, operator)
    elif operator == "xor":
        perceptron(xor_features, xor_labels, operator)

    elif operator == "exit":
        break
    else:
        print("Invalid Input.\nTry Again!")
```

RESULT:

Thus, the algorithm can correctly classify the data points with a linear decision boundary. We prepared the data, initialized the weights, and trained & tested the model. All the results were verified successfully.

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...
```

```
In [1]:         'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester VI/
19CSE456 - Neural Networks and Deep Learning/Lab/Practical 2 - Perceptron Algorithm for a
Binary Classification Problem/19CSE456_Practical_2_Code.py'      ='E:/Plan B/Amrita Vishwa
Vidyapeetham/Subject Materials/Semester VI/19CSE456 - Neural Networks and Deep Learning/
Lab/Practical 2 - Perceptron Algorithm for a Binary Classification Problem'

Weight:  [0.65311937 0.39002368]
Bias:  0.11706019576186966
Alpha:  0.1
Epoch:  1000


Enter Logical Operator (and, or, xor, exit): and

Epoch Value:  1
0  and  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.5531193686335357 & w2: 0.29002368121292665
)
0  and  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.5531193686335357 & w2: 0.29002368121292665
)
1  and  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.5531193686335357 & w2: 0.29002368121292665
)
1  and  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.5531193686335357 & w2: 0.29002368121292665
)

Epoch Value:  2
0  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.19002368121292665
)
0  and  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.19002368121292665
)
1  and  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.19002368121292665
)
1  and  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.19002368121292665
)

Epoch Value:  3
0  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
0  and  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)

Epoch Value:  4
0  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
```

```
)
0  and  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)


Enter Logical Operator (and, or, xor, exit): and

Epoch Value:  1
0  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
0  and  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)
1  and  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.35311936863353577 & w2: 0.19002368121292665
)


Enter Logical Operator (and, or, xor, exit): or

Epoch Value:  1
0  or  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
0  or  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)

Epoch Value:  2
0  or  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
0  or  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)


Enter Logical Operator (and, or, xor, exit): or

Epoch Value:  1
0  or  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
0  or  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
)
1  or  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.45311936863353575 & w2: 0.29002368121292665
```

```
)

Enter Logical Operator (and, or, xor, exit): exit

In [2]:
```

```
-0.03690515890115206 )

Epoch Value:  990
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)

Epoch Value:  991
0  xor  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
0  xor  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )

Epoch Value:  992
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)

Epoch Value:  993
0  xor  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
0  xor  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )

Epoch Value:  994
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)

Epoch Value:  995
0  xor  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
```

```
0  xor  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )

Epoch Value:  996
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)

Epoch Value:  997
0  xor  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
0  xor  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )

Epoch Value:  998
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)

Epoch Value:  999
0  xor  0 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
0  xor  1 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  0 -> Actual: 1 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )
1  xor  1 -> Actual: 0 ; Predicted: 1  ( w1: 0.02991371004733384 & w2:
-0.03690515890115206 )

Epoch Value:  1000
0  xor  0 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
0  xor  1 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  0 -> Actual: 1 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
)
1  xor  1 -> Actual: 0 ; Predicted: 0  ( w1: 0.12991371004733385 & w2: 0.06309484109884794
```

```
)

Enter Logical Operator (and, or, xor, exit): exit

In [3]:
```