SHRI SAI ARAVIND R
212223040197
EX. NO.5
DATE:

Implementation of XOR using RBF

Aim:
To implement a XOR gate classification using Radial Basis Function Neural Network.

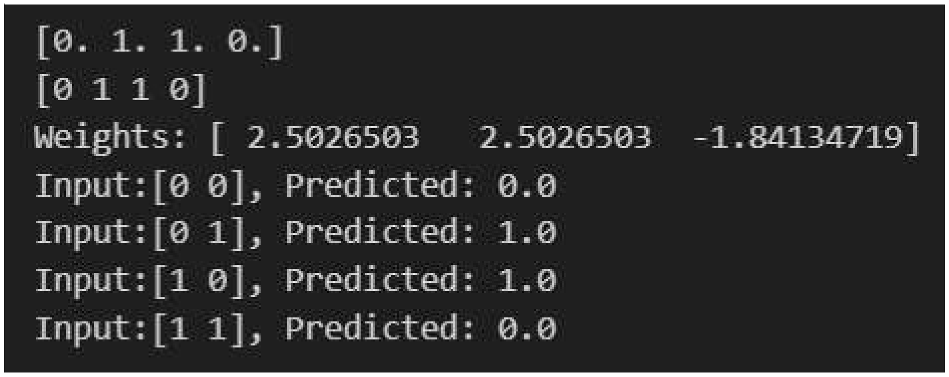
Theory:
Exclusive or is a logical operation that outputs true when the inputs differ.For the XOR gate, the TRUTH table will be as follows XOR truth table
XOR is a classification problem, as it renders binary distinct outputs. If we plot the INPUTS vs OUTPUTS for the XOR gate, as shown in figure below
The graph plots the two inputs corresponding to their output. Visualizing this plot, we can see that it is impossible to separate the different outputs (1 and 0) using a linear equation. A Radial Basis Function Network (RBFN) is a particular type of neural network. The RBFN approach is more intuitive than MLP. An RBFN performs classification by measuring the input's similarity to examples from the training set. Each RBFN neuron stores a "prototype", which is just one of the examples from the training set. When we want to classify a new input, each neuron computes the Euclidean distance between the input and its prototype. Thus, if the input more closely resembles the class A prototypes than the class B prototypes, it is classified as class A ,else class B. A Neural network with input layer, one hidden layer with Radial Basis function and a single node output layer (as shown in figure below) will be able to classify the binary data according to XOR output.

ALGORITHM:
Step 1: Initialize the input vector for you bit binary data
Step 2: Initialize the centers for two hidden neurons in hidden layer
Step 3: Define the non- linear function for the hidden neurons using Gaussian RBF
Step 4: Initialize the weights for the hidden neuron
Step 5 : Determine the output function as Y=W1*φ1 +W1 *φ2
Step 6: Test the network for accuracy
Step 7: Plot the Input space and Hidden space of RBF NN for XOR classification.

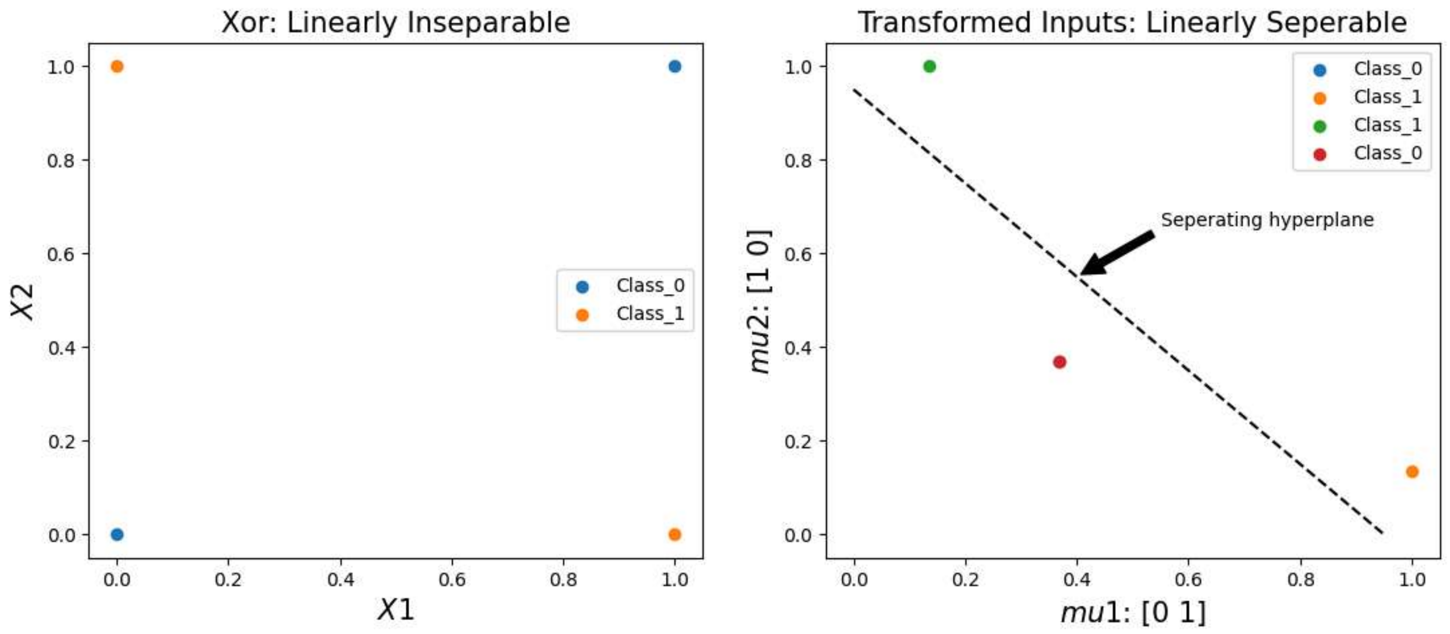
PROGRAM:
```
import numpy as np
import matplotlib.pyplot as plt


def gaussian_rbf(x, landmark, gamma=1):
    return np.exp(-gamma * np.linalg.norm(x - landmark)**2)

def end_to_end(X1, X2, ys, mu1, mu2):
    from_1 = [gaussian_rbf(np.array([X1[i], X2[i]]), mu1) for i in range(len(X1))]
    from_2 = [gaussian_rbf(np.array([X1[i], X2[i]]), mu2) for i in range(len(X1))]
    # plot

    plt.figure(figsize=(13, 5))
    plt.subplot(1, 2, 1)
    plt.scatter((x1[0], x1[3]), (x2[0], x2[3]), label="Class_0")
    plt.scatter((x1[1], x1[2]), (x2[1], x2[2]), label="Class_1")
    plt.xlabel("$X1$", fontsize=15)
    plt.ylabel("$X2$", fontsize=15)
    plt.title("Xor: Linearly Inseparable", fontsize=15)
    plt.legend()



    plt.subplot(1, 2, 2)
    plt.scatter(from_1[0], from_2[0], label="Class_0")
    plt.scatter(from_1[1], from_2[1], label="Class_1")
    plt.scatter(from_1[2], from_2[2], label="Class_1")
    plt.scatter(from_1[3], from_2[3], label="Class_0")
    plt.plot([0, 0.95], [0.95, 0], "k--")
    plt.annotate("Seperating hyperplane", xy=(0.4, 0.55), xytext=(0.55, 0.66),
                 arrowprops=dict(facecolor='black', shrink=0.05))
    plt.xlabel(f"$mu1$: {(mu1)}", fontsize=15)
    plt.ylabel(f"$mu2$: {(mu2)}", fontsize=15)
    plt.title("Transformed Inputs: Linearly Seperable", fontsize=15)
    plt.legend()
    # solving problem using matrices form
    # AW = Y
    A = []
    for i, j in zip(from_1, from_2):
        temp = []
        temp.append(i)
        temp.append(j)
        temp.append(1)
        A.append(temp)

    A = np.array(A)

    #Type here
    W = np.linalg.inv(A.T.dot(A)).dot(A.T).dot(ys)
    print(np.round(A.dot(W)))
    print(ys)
    print(f"Weights: {W}")
    return W

def predict_matrix(point, weights):
    #Type your code here
    gaussian_rbf_0 = gaussian_rbf(point, mu1)
    gaussian_rbf_1 = gaussian_rbf(point, mu2)
    A = np.array([gaussian_rbf_0, gaussian_rbf_1, 1])
    return np.round(A.dot(weights))

x1 = np.array([0, 0, 1, 1])
x2 = np.array([0, 1, 0, 1])
ys = np.array([0, 1, 1, 0])
```

```
# centers
mu1 = np.array([0, 1])
mu2 = np.array([1, 0])

w = end_to_end(x1, x2, ys, mu1, mu2)


print(f"Input:{np.array([0, 0])}, Predicted: {predict_matrix(np.array([0, 0]), w)}")
print(f"Input:{np.array([0, 1])}, Predicted: {predict_matrix(np.array([0, 1]), w)}")
print(f"Input:{np.array([1, 0])}, Predicted: {predict_matrix(np.array([1, 0]), w)}")
print(f"Input:{np.array([1, 1])}, Predicted: {predict_matrix(np.array([1, 1]), w)}")
```

OUTPUT:



```
[0. 1. 1. 0.]
[0 1 1 0]
Weights: [ 2.5026503   2.5026503  -1.84134719]
Input:[0 0], Predicted: 0.0
Input:[0 1], Predicted: 1.0
Input:[1 0], Predicted: 1.0
Input:[1 1], Predicted: 0.0
```



Result:

Thus , a Radial Basis Function Neural Network is implemented to classify XOR data.