Batch: A2 **Roll No.: 16010121045**

Experiment No. 02

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To implement XOR LOGIC using perceptron network. .

Objective: To implement classifier using Multi-layer perceptron network for XOR logic of 2 inputs.

Expected Outcome of Experiment:

CO2 : Analyze various neural network architectures

Books/ Journals/ Websites referred:

- .

Pre Lab/ Prior Concepts:

Perceptron Model

The perceptron is a fundamental concept in artificial neural networks and machine learning. It was introduced in the late 1950s by Frank Rosenblatt. The perceptron is a simplified model of a biological neuron and serves as a building block for more complex neural network architectures. At its core, a perceptron takes multiple input signals, each with an associated weight, and produces an output based on a weighted sum of these inputs. The perceptron's output is then typically passed through an activation function to produce the final output of the perceptron. Mathematically, the output of a perceptron can be represented as follows:

Output = ActivationFunction(WeightedSum + Bias)

Where:
WeightedSum is the sum of the products of input values and their corresponding weights.
Bias is an additional constant term added to the weighted sum.

ActivationFunction is a non-linear function that determines whether the perceptron should "fire" (produce an output signal) based on the weighted sum.
The perceptron model can be trained using a learning algorithm, such as the perceptron learning rule, which adjusts the weights and bias in order to correctly classify input data. However, the perceptron has limitations; it can only classify linearly separable data.

Linear separability

Linear separability is a concept in linear algebra and machine learning that relates to whether two classes of data points can be separated by a straight line (in 2D), a hyperplane (in higher dimensions), or a linear decision boundary in general. In the context of binary classification, if it's possible to draw a line, plane, or hyperplane that can completely separate the data points of one class from those of the other class, the data is considered linearly separable.

Linear separability is important because it defines the boundary between classes in a way that simplifies classification tasks. In the context of the perceptron model, it means that the perceptron can accurately classify the data if the data is linearly separable. If the data is not linearly separable, the perceptron learning algorithm might struggle to converge to a solution.

To determine if data is linearly separable, you typically need to analyze the distribution of data points and assess whether a linear decision boundary can cleanly separate the two classes. If the data is not linearly separable, more complex models or techniques such as kernel methods may be required to accurately classify the data.

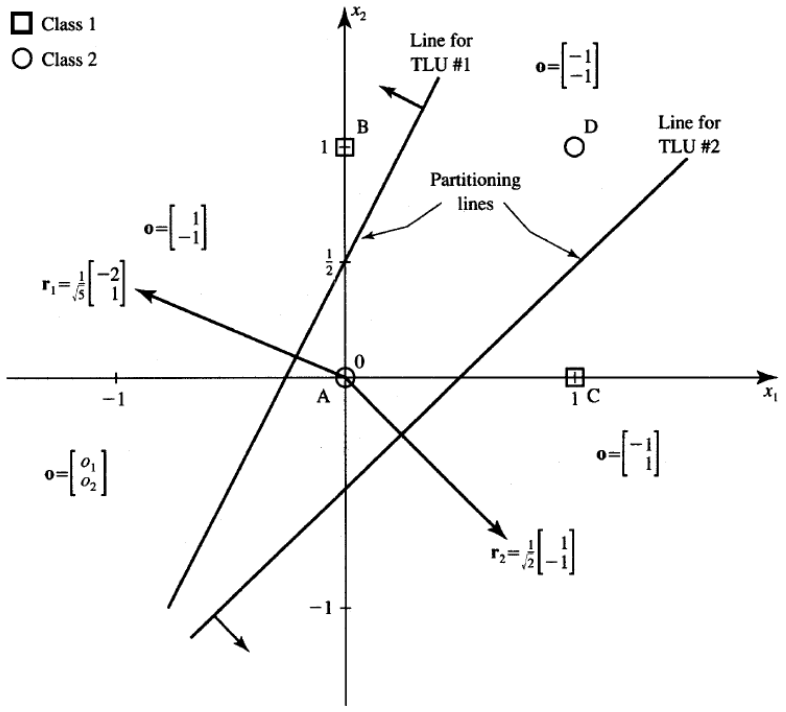
In summary, the perceptron model is a foundational concept in neural networks and machine learning, while linear separability refers to the property of data being separable by a linear decision boundary. Understanding these concepts lays the groundwork for exploring more advanced topics in machine learning and artificial intelligence.

Design of Classifier using Multi-layer Perceptron model for XOR logic of 2 inputs (Refer Zurada 4.1 page no `68)

Truth table for XOR logic

x_1	x_2	Output
0	0	1
0	1	−1
1	0	−1
1	1	1

Perceptron (TLU #1 and TLU +2) for first layer using bipolar activation function



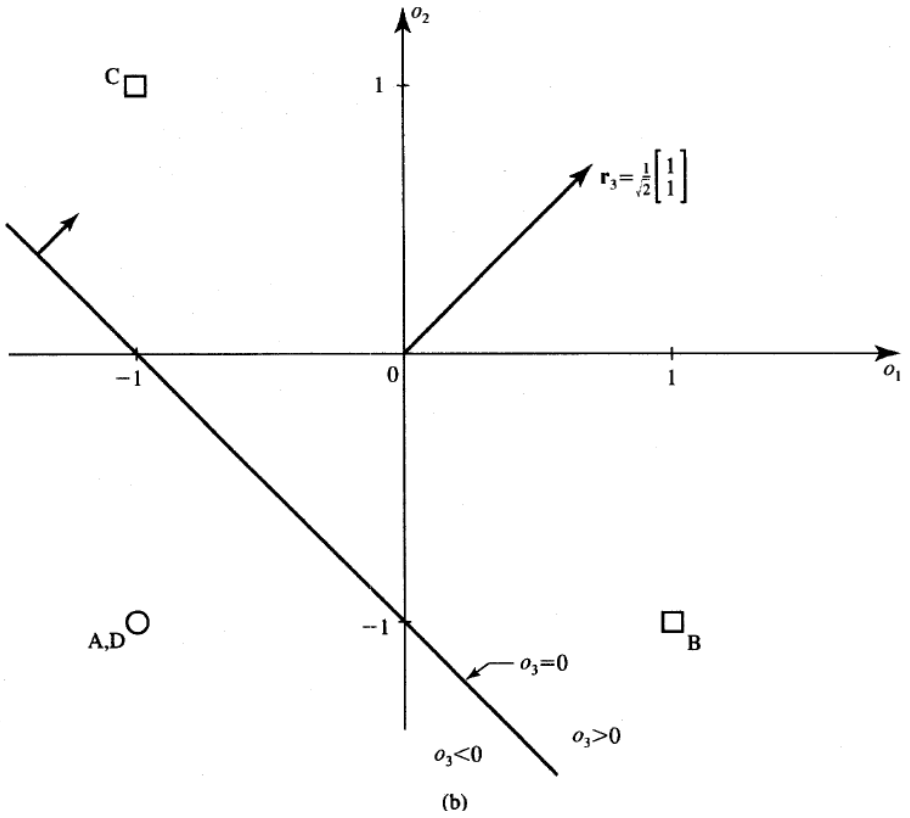
Two decision lines having equations using arbitrary selected partitioning as shown in above figure.

$$-2x_1 + x_2 - \frac{1}{2} = 0$$
$$x_1 - x_2 - \frac{1}{2} = 0$$

Mapping performed by first layer perceptron's TLU#1 and TLU#2

$$o_1 = \text{sgn}\left(-2x_1 + x_2 - \frac{1}{2}\right)$$
$$o_2 = \text{sgn}\left(x_1 - x_2 - \frac{1}{2}\right)$$

Final Output layer perceptron TLU#3



(b)

The decision line represents the equation

$$o_1 + o_2 + 1 = 0$$

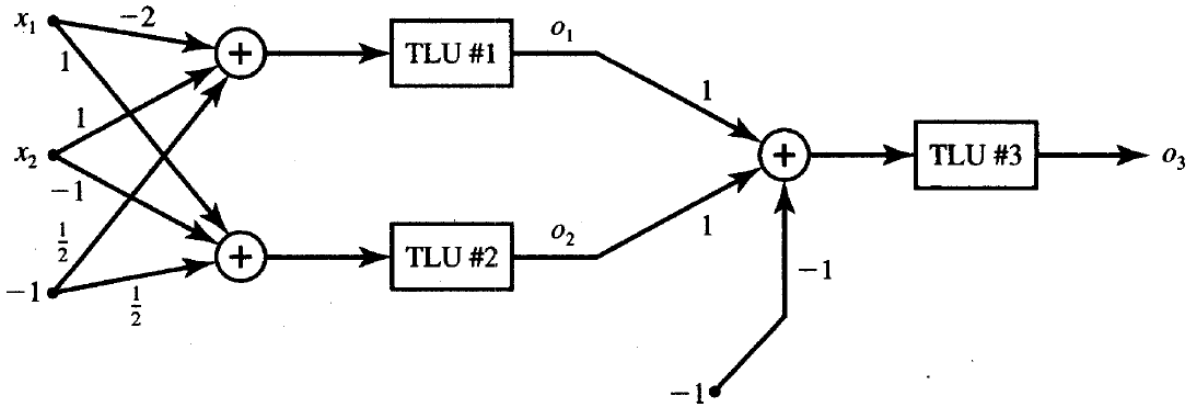
Mapping performed by output perceptron

The TLU #3 implements the decision o_3 as

$$o_3 = \text{sgn}(o_1 + o_2 + 1)$$

Classification summary table

Symbol	Pattern Space x_1	Pattern Space x_2	Image Space o_1	Image Space o_2	TLU #3 Input $o_1 + o_2 + 1$	Output Space o_3	Class Number
A	0	0	−1	−1	−	−1	2
B	0	1	1	−1	+	+1	1
C	1	0	−1	1	+	+1	1
D	1	1	−1	−1	−	−1	2



Classifier using Multi-layer Perception Network for implementing XOR logic for 2 inputs.

Code:

```
import numpy as np

def step_function(value):
    if value >= 0:
        return 1
    else:
        return 0

def perceptron(input_data, weights, bias):
    weighted_sum = np.dot(weights, input_data) + bias
    output = step_function(weighted_sum)
    return output

def NOT_logic(input_data):
    not_weight = -1
    not_bias = 0.7
    return perceptron(input_data, not_weight, not_bias)

def AND_logic(input_data):
    and_weights = np.array([1, 1])
    and_bias = -1.7
    return perceptron(input_data, and_weights, and_bias)

def OR_logic(input_data):
    or_weights = np.array([1, 1])
```

```
    or_bias = -0.5
    return perceptron(input_data, or_weights, or_bias)

def NAND_logic(input_data):
    nand_weights = np.array([-1, -1])
    nand_bias = 1.5
    return perceptron(input_data, nand_weights, nand_bias)

def XOR_logic(input_data):
    nand_output = NAND_logic(input_data)
    or_output = OR_logic(input_data)
    combined_input = np.array([nand_output, or_output])
    xor_output = AND_logic(combined_input)
    return xor_output

def test_logic_function(logic_func, test_input):
    result = logic_func(test_input)
    return result

num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
test_input = [num1, num2]

xor_result = test_logic_function(XOR_logic, test_input)
print("XOR({}, {}) = {}".format(test_input[0], test_input[1], xor_result))
```

Output:

```
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exp2.py"
Enter the first number: 1
Enter the second number: 1
XOR(1, 1) = 0
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exp2.py"
Enter the first number: 0
Enter the second number: 1
XOR(0, 1) = 1
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exp2.py"
Enter the first number: 0
Enter the second number: 0
XOR(0, 0) = 0
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exp2.py"
Enter the first number: 1
Enter the second number: 0
XOR(1, 0) = 1
```

Conclusion: Thus, we have successfully implemented classifier using MLP for linearly non-separable XOR functions of 2 inputs

Post Lab Descriptive Questions :

1. Why is XOR-logic function being Linearly notseparable?

The XOR (exclusive OR) function is linearly non-separable because it cannot be represented using a single linear decision boundary. In a two-dimensional input space, where the two input features are plotted on the axes, XOR's truth table has the following structure:

Truth Table		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

If you try to draw a single straight line (a linear decision boundary) to separate the points representing output 1 from output 0, you'll find that it's impossible. The points are interspersed in such a way that no single straight line can separate them into two distinct regions.

Linearly separable problems are those where you can draw a single straight line to classify points of different classes. In contrast, XOR requires a nonlinear decision boundary, which cannot be achieved using a single-layer perceptron with linear activation functions.

2. Design a classifier using MLP perceptron model for implementing NOT XOR logic

To implement the NOT XOR logic using a Multi-Layer Perceptron (MLP), we need to use at least one hidden layer with nonlinear activation functions. Let's design a simple MLP with one hidden layer to achieve NOT XOR logic.
In this example, we use the sigmoid activation function for both the hidden layer and the output layer. The hidden layer with appropriate weights and biases effectively transforms the input space into a form where the NOT XOR logic can be separated linearly. The output layer then produces the final result.

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def perceptron(input_data, weights, bias):
    weighted_sum = np.dot(weights, input_data) + bias
    output = sigmoid(weighted_sum)
    return output

def mlp_xor_not(input_data):
    # Hidden layer weights and biases
    hidden_weights = np.array([[20, 20], [-20, -20]])
    hidden_bias = np.array([-10, 30])

    # Output layer weights and bias
    output_weights = np.array([[-20, 20]])
```

```
    output_bias = np.array([10])

    # Calculate hidden layer outputs
    hidden_output = perceptron(input_data, hidden_weights, hidden_bias)

    # Calculate final output using hidden layer outputs
    final_output = perceptron(hidden_output, output_weights, output_bias)

    return np.round(final_output)

num1 = int(input("Enter the number: "))
test_input = np.array([num1])
xor_not_result = mlp_xor_not(test_input)
print("NOT XOR({}) = {}".format(test_input[0], xor_not_result))
```

Date: _____ **Signature of faculty in-charge**