Practical No.10

Title: Implement a single neural network and test for different logic gates

Objective

The objective of this practical is to implement a **single-layer neural network** and test it for different **logic gates**, including **AND, OR, NAND, NOR, and XOR**. The neural network will be trained using a suitable learning algorithm and tested for correctness.

Introduction to Neural Networks

A **Neural Network** is a computational model inspired by the human brain. It consists of interconnected units called **neurons**, which process information and learn patterns from data.

Basic Structure of a Single-Layer Neural Network

A single-layer neural network (also known as a **Perceptron**) consists of:

1. **Input Layer** – Takes binary inputs (e.g., A and B for logic gates).
2. **Weights and Bias** – Each input is multiplied by a weight, and a bias is added.
3. **Summation Function** – Computes the weighted sum of inputs:

Net=(w1×x1)+(w2×x2)+bNet = (w_1 \times x_1) + (w_2 \times x_2) + bNet=(w1×x1)+(w2×x2)+b

4. **Activation Function** – Applies a threshold to determine output:

Output=f(Net)Output = f(Net)Output=f(Net)

where fff is a step function (e.g., Heaviside function for binary classification).

Activation Function Used

The most commonly used activation function for binary classification is the **step function**:

f(x)={1,if x≥00,if x<0f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}f(x)={1,0,if x≥0if x<0

Logic Gates and Their Truth Tables

Logic gates take **binary inputs (0 or 1)** and produce a **binary output** based on logical operations.

A	**B**	**AND**	**OR**	**NAND**	**NOR**	**XOR**
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

- **AND, OR, NAND, and NOR** gates are **linearly separable**, meaning they can be learned using a **single-layer Perceptron**.
- **XOR is not linearly separable** and requires a **multi-layer network** to be implemented correctly.

Implementation Steps

Step 1: Initialize Network Parameters

- Define **input values** (A and B) for logic gates.
- Initialize **weights** and **biases** randomly.

Step 2: Compute Output Using Perceptron Model

- Calculate the weighted sum of inputs.
- Apply the **activation function** to determine the output.

Step 3: Train the Neural Network (Learning Process)

- Use a **Supervised Learning Algorithm** like **Perceptron Learning Rule**:
    - Adjust weights based on errors using:

wnew=wold+η×(Target−Output)×Inputw_{new} = w_{old} + \eta \times (Target - Output) \times Inputwnew=wold+η×(Target−Output)×Input

where η\etaη is the learning rate.

- Iterate over training data until the model classifies all cases correctly.

Step 4: Test the Neural Network

- Provide different input values.
- Compare the predicted output with the expected output from truth tables.

Step 5: Evaluate Model Performance

- Check if the model correctly classifies each logic gate.
- If incorrect, update weights and retrain.

Expected Output

The program will output:

- **Trained weights and bias values** for each logic gate.
- **Predicted outputs** for test inputs.
- **Correct classification for AND, OR, NAND, and NOR gates**.
- **Incorrect classification for XOR gate (since a single-layer Perceptron cannot learn XOR)**.

Conclusion

This practical demonstrates the implementation of a **single-layer Perceptron** for **logic gates**. The model was trained and tested for different logic functions.