**Challenge 6 – Perceptron with Sigmoid Activation**

**1. Introduction**

In this challenge, we implement a basic perceptron (a simple artificial neuron) that uses the sigmoid activation function. The perceptron is trained using the perceptron learning rule to learn binary logic functions. Specifically, we aim to train it to model the NAND and XOR gates.

The goal is to understand how a simple neural unit can model linearly separable logic gates, and why more complex architectures (such as multi-layer perceptrons) are needed to model non-linear functions like XOR.

**2. Theory**

**Sigmoid Function**

The sigmoid activation function is defined as

A black background with white text

AI-generated content may be incorrect.

It maps input values to a range between 0 and 1, making it suitable for binary classification.

**Derivative of Sigmoid**

****

This derivative is used in backpropagation for updating weights.

**Perceptron Learning Rule**

Weights and bias are as follows:

A black background with white text

AI-generated content may be incorrect.  
Where:

* η is the learning rate
* error = target - output
* σ'(output) is the derivative of sigmoid

**Linear Separability**

A logic function is linearly separable if a straight line can separate its output classes. NAND is linearly separable; XOR is not.

**3. Task 1: Implementing a Simple Perceptron**

We implemented a perceptron class with the following capabilities:

* Random initialization of weights and bias
* Forward pass using the sigmoid function
* Training using the perceptron learning rule

This perceptron accepts two inputs and outputs a value between 0 and 1.

A number of numbers and symbols

AI-generated content may be incorrect.

**4. Task 2a: Training on NAND**

**Dataset:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **NAND** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Output after training:**

**A black and white text with numbers

AI-generated content may be incorrect.**

**Observation:**

The perceptron successfully learned the NAND function, producing accurate outputs.

**5. Task 2b: Training on XOR (Single-Layer Perceptron)**

**Dataset:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **XOR** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Output after training:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Observation:**

The perceptron failed to learn the XOR gate. This is because XOR is not linearly separable and a single-layer perceptron cannot model such relationships.

**6. Bonus: XOR with Multi-Layer Perceptron**

To solve XOR, we implemented a multi-layer perceptron with:

* 2 input neurons
* 2 hidden neurons with sigmoid activation
* 1 output neuron with sigmoid activation

A screenshot of a computer

AI-generated content may be incorrect.

This model is capable of learning non-linear functions by introducing a hidden layer that adds complexity and non-linearity to the model.

**Output after training:**

**A number of numbers on a white background

AI-generated content may be incorrect.**

**Observation:**

The MLP successfully learned the XOR function, demonstrating that hidden layers are essential for solving non-linear problems.

**7. Conclusion**

This challenge demonstrated how a simple perceptron can effectively model linearly separable logic gates like NAND using sigmoid activation and the perceptron learning rule. However, it also highlighted the limitations of single-layer perceptrons in learning non-linearly separable functions like XOR. To overcome this, a multi-layer perceptron was implemented, which successfully learned the XOR function by adding hidden layers and non-linear activation.

**8. Appendix**

* **Code files:** nand\_perceptron.py, perceptron.py, xor\_perceptron.py, xor\_mlp.py
* **Tools used:** Python, Jupyter