**AIN SAHMS UNIVERSITY**

**FACULTY OF ENGINEERING**

**Computational Intelligence (CSE473s)**

**Fall 2025**

# Group 25 Submission part 1

# Project Title: Build Your Own Neural Network Library & Advanced

# Part 1: Library Implementation, Initial Test and Validation (XOR Problem)



| | |
|---|---|
| **Mina Ashraf Rizk Habib** | **2100447** |
| **Mario Medhat Roshdy** | **2100513** |
| **David Gerges Ishak** | **2100834** |
| **Emannuel Geroge Chawky** | **2100073** |



**Submitted to: DR. Hossam El-Din Hassan Abd El Munim**

**Eng. Abdallah Mohamed Mahmoud Ahmed Awdallah**

**Date: December 05, 2025**

# Table of Contents



# Table of Figures

# 1.Objectives

The primary objective of this phase was to develop a modular neural network library from scratch using **Python and NumPy**. To validate the mathematical correctness of the forward and backward propagation algorithms, the library was tested on the classic **XOR problem**, a non-linear classification task that a simple linear model cannot solve.

# 2.Library Architecture

The library was designed with a modular object-oriented approach:

- **Layer Abstraction:** A base Layer class enforces the implementation of forward () and backward () methods.
- **Dense Layer:** Implements fully connected layers computing $Y = XW + b$. It calculates gradients $A = \frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ for optimization.
- **Activations:** Implemented as separate layers to allow flexibility.
    - **Tanh:** Used for hidden layers to center data around zero.
    - **Sigmoid:** Used for the output layer to map predictions to a probability range (0, 1).
- **Optimization:** An **SGD (Stochastic Gradient Descent)** optimizer was implemented to update weights based on the calculated gradients and a learning rate ($\boldsymbol{\eta}$).
- **Loss Function: Mean Squared Error (MSE)** was used to quantify the error between predictions and targets.

# 3.Experimental Setup (XOR Problem)

To solve the XOR function, we constructed a Multi-Layer Perceptron (MLP) with the following architecture, as specified in the project requirements:

- **Input Layer:** 2 Neurons (representing the two binary inputs).
- **Hidden Layer:** 4 Neurons with **Tanh** activation.
- **Output Layer:** 1 Neuron with **Sigmoid** activation.

**Hyperparameters:**

- **Learning Rate:** 1.0
- **Epochs:** 2000
- **Loss Function:** MSE

# 4. Results and Analysis

## 4.1 Training Performance

The network was trained for 2000 epochs. As shown in Figure 1, the Mean Squared Error (MSE) drastically decreased, converging near zero. Simultaneously, the accuracy stabilized at 100%.

Training metrics showing the convergence of Loss and the improvement of Accuracy over time.

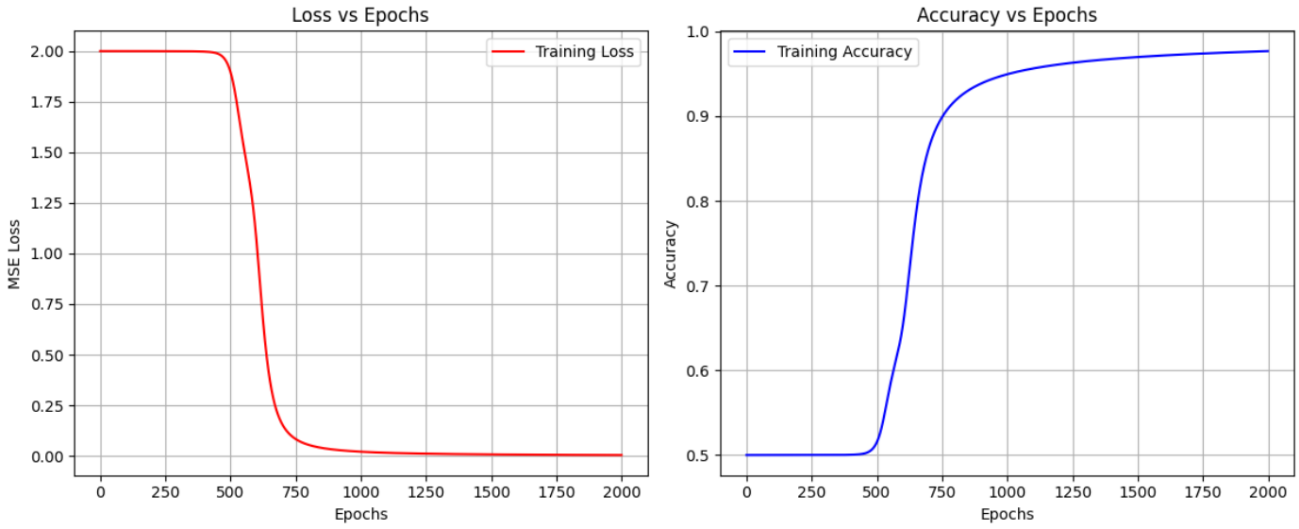


*Figure 1 Training metrics*

## 4.2 Decision Boundary

To visualize the non-linearity of the solution, we plotted the decision boundary. **Figure 2** demonstrates that the network successfully learned a curved boundary to separate the classes (0 vs 1), which proves the effectiveness of the hidden layer and non-linear activations.

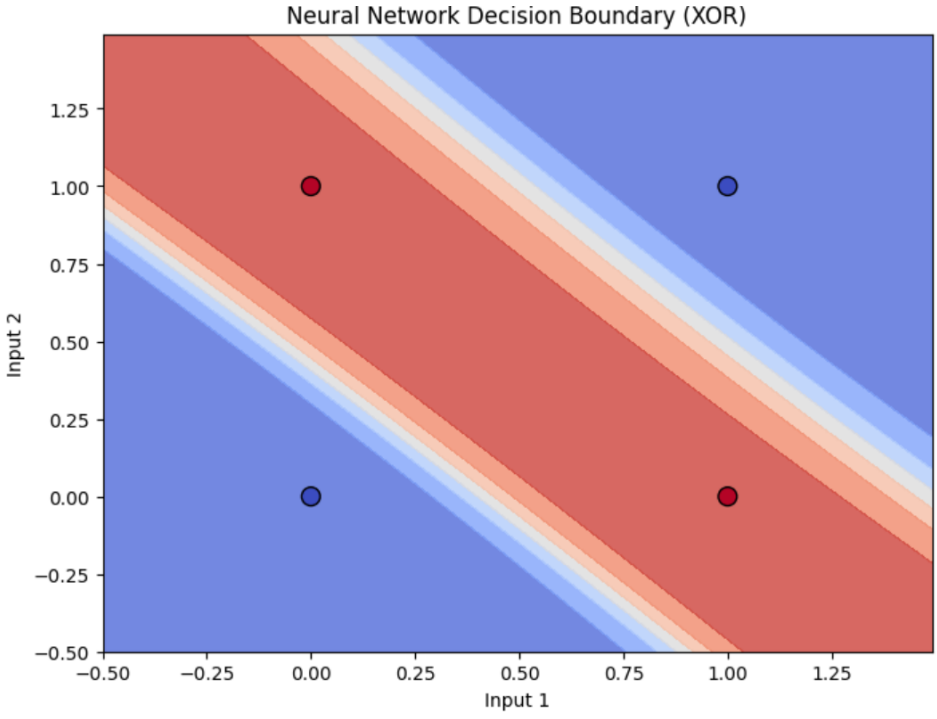


*Figure 2 The learned decision boundary separating the XOR classes.*

### 4.3 Final Predictions

After training, the model was evaluated on all 4 possible inputs. The results are perfect, meeting the validation goal:

| Input A | Input B | Target | Predicted Probability | Rounded Output |
|---|---|---|---|---|
| 0 | 0 | **0** | *0.0229* | **0** |
| 0 | 1 | **1** | *0.9764* | **1** |
| 1 | 0 | **1** | *0.9803* | **1** |
| 1 | 1 | **0** | *0.0266* | **0** |

# 5. Results and Analysis using TensorFlow

Training metrics on TensorFlow showing the convergence of Loss and the improvement of Accuracy over time are coming at a much faster rate than our model.

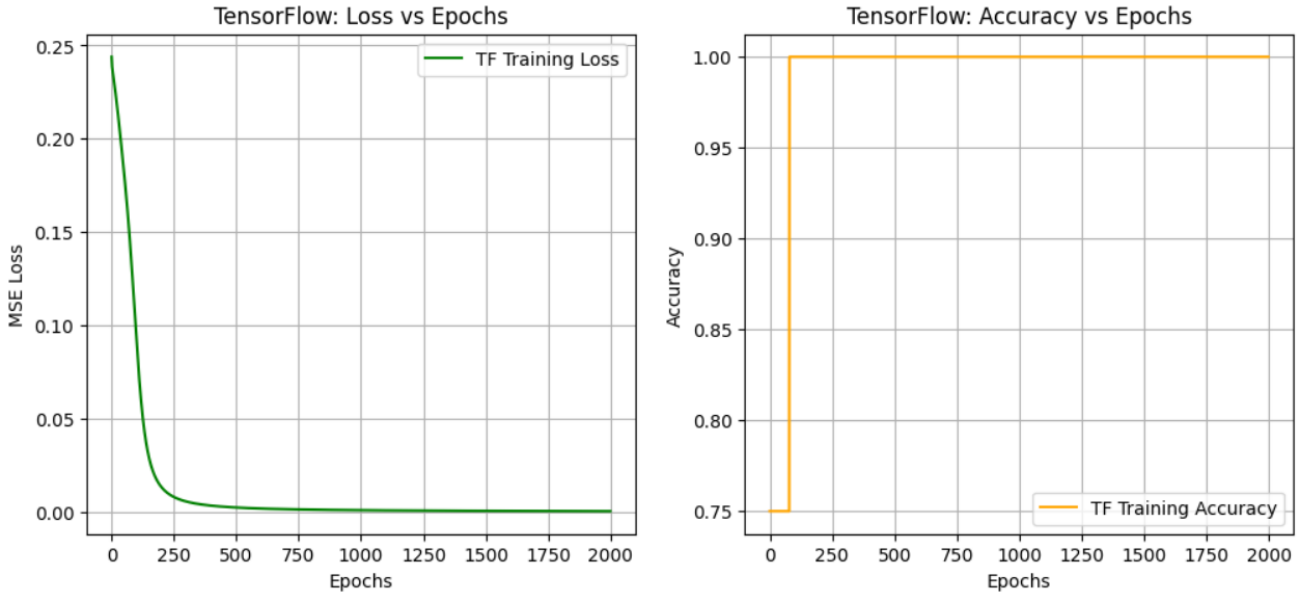


*Figure 3 Training metrics on TensorFlow*

And it gets almost the same predative answer only on 1 alteration

Final TensorFlow Predictions:

**1/1** ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 250ms/step

Input: [0. 0.], Pred: 0.0199, Rounded: 0

Input: [0. 1.], Pred: 0.9801, Rounded: 1

Input: [1. 0.], Pred: 0.9837, Rounded: 1

Input: [1. 1.], Pred: 0.0221, Rounded: 0

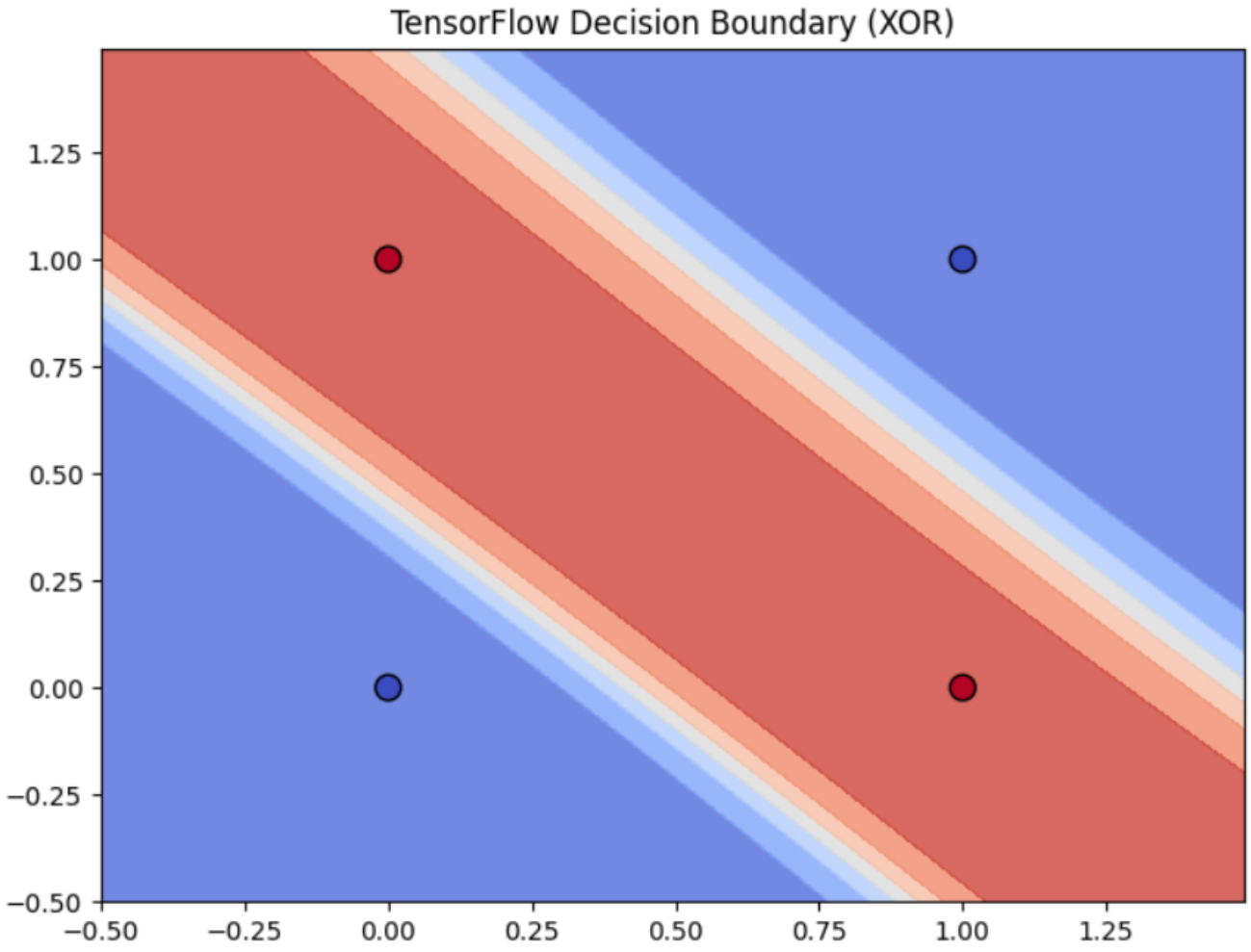


*Figure 4 The learned decision boundary separating the XOR classes using the TensorFlow library.*

# 6. Conclusion

The successful convergence on the XOR problem validates that the core components of the library specifically matrix multiplication in Dense layers and the chain rule implementation in backward propagation are mathematically correct. The library is now ready for complex tasks like image reconstruction.

# 7. GitHub repo

https://github.com/minaashraf9822/Computational-Intelligence-project-Fall-2025

# 8. Appendix

## 8.1 Code Structure

```
🗁 Repository StructureThe project follows the mandatory structure required by the course: Plaintext/
├── lib/ # Custom Neural Network Library
│ ├── **init**.py
│ ├── layers.py # Dense Layer implementation
│ ├── activations.py # ReLU, Sigmoid, Tanh
│ ├── losses.py # MSE Loss & Prime
│ ├── optimizer.py # SGD Optimizer
│ ├── network.py # Network class (train/predict loops)
│ ├── encoder.py # Encoder Architecture helper
│ ├── decoder.py # Decoder Architecture helper
│ ├── data_loader.py # MNIST loading & preprocessing
│ └── training_history.py # Visualization tools
│
├── notebooks/
│ └── project_demo.ipynb # Main Demo (XOR, Autoencoder, SVM) [cite: 101]
│
├── report/
│ └── project_report.pdf # Detailed Analysis & Results [cite: 95]
│
├── .gitignore
├── README.md
└── requirements.txt
```