AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Computational Intelligence CSE473s
Spring 2025 / Mechatronics Senior-2

Final Project

[Neural Network Library]

– Team – 18

Name	ID
Amr Khaled Taha	2101427
Mostafa Samy Mohammed	2100349
Mohamed Tarek Abdelwahab	2100287
Omar Mohamed Fathy	2100503

Submitted to:

Prof. Dr. Hossam El din Hassan

Eng. Abdallah Awdallah

Introduction

This project implements a neural network library **from scratch** using NumPy. Part 1 focuses on developing the **core training pipeline**, including:

- Fully Connected (`Dense`) layers
- Nonlinear Activation functions (ReLU, Sigmoid, Tanh, Softmax)
- Loss computation using **Mean Squared Error** (`MSELoss`)
- Gradient-based weight updates using **Stochastic Gradient Descent** (`SGDOptimizer`)
- A `Sequential` model class to combine all components
- Experimental validation using the **XOR problem**

This implementation provides a foundational understanding of forward propagation, backward propagation, and parameter updates — without relying on deep learning frameworks.

System Architecture & Design

Overall Workflow

```
Input → Sequential.forward → MSELoss → MSELoss.backward → Sequential.backward
→ SGDOptimizer(step)
```

Modular Design Strategy

Component	Responsibility
`Sequential`	Manages layers, full training loop (`fit`)
`Dense`	Trainable linear transformation (stores `W`, `b`, `dW`, `db`)
Activation Functions	Apply nonlinearity, no trainable parameters
`MSELoss`	Computes loss and its gradient w.r.t model output
`SGDOptimizer`	Updates parameters using gradients

The separation allows future extension:

- More optimizers (Momentum, Adam)
- Additional loss functions
- Regularization
- More layer types

Implemented Components

Dense Layer

- Trainable weights: `W` and b
- Uses scaled random initialization: `np.random.randn() * scale`
- Stores gradients: `dW`, db

Forward [Z = XW + b]

Backward [dW = XT dZ,\quad db = \sum(dZ),\quad dX = dZ WT]

Activation Layers

Activation	Derivative
ReLU	$(1_{x>0})$
Sigmoid	$(y(1-y))$
Tanh	(1 − y^2)
Softmax	Used later with cross-entropy

- Non-trainable (only modify the gradient).

`MSELoss` class

Used for XOR regression-style learning.

[L = \frac{1}{N} \sum (y - \hat{y})^2] [\frac{\partial L}{\partial \hat{y}} = \frac{2}{N}(\hat{y} - y)]

`SGDOptimizer` class

Updates trainable parameters:

[W := W - \eta \cdot dW] [b := b - \eta \cdot db]

Where (\eta) = learning rate.

`Sequential` class

Responsibilities:

- Store ordered layers
- `forward()` → inference
- `backward()` → propagate gradients

- `fit()` → complete training loop
    1. Forward pass
    2. Computing loss
    3. Backward pass
    4. Optimizer weight update

Result: **Simple high-level API** for training models.

Experiment: XOR Problem

Dataset

Input (x1, x2)	Target (y_true)
[0,0]	0
[0,1]	1
[1,0]	1
[1,1]	0

- XOR is **non-linearly separable**
- Validates backprop & nonlinear learning

Model Architecture

Layer	Scaling factor (Parameters initialization)	Details
Dense layer (2 -> 4)	1.0	Hidden layer
Tanh	–	Nonlinear activation
Dense layer (4 -> 1)	1.0	Output layer
Sigmoid	–	Output activation

- Loss function: [**MSELoss]**.
- Optimizer: [**SGDOptimizer]** with learning rate (η) = 1.0, and epochs = 10,000.

Results

Loss Curve

Insert your loss plot here from notebook Example caption:

> The loss decreases over training, confirming correct backward propagation.

Final Predictions

Input	Target	Predicted	Rounded
[0,0]	0	0.00321	0
[0,1]	1	0.99170	1
[1,0]	1	0.99166	1
[1,1]	0	0.00996	0

The model successfully learned XOR mapping, achieving correct classification.

Observations & Notes

- With **MSE + Sigmoid**, learning can be slower due to saturation
- Increasing:
  - Hidden layer size
  - Learning rate Can improve convergence
- The pipeline correctly computes gradients and updates weights

This confirms our framework works end-to-end.

Conclusion

- Fully functional neural network training library implemented
- Verified using XOR experiment
- Solid foundation for Part 2 improvements:
  - Softmax + Cross-Entropy for classification
  - More optimizers
  - Model evaluation metrics
  - Better weight initialization
  - Saving/loading models

The implementation demonstrates fundamental understanding of neural networks and backpropagation.