AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Senior-2 Mechatronics

CSE473s: Computational Intelligence

Fall 2025

Part 1 Project Report

Library Implementation, Initial Test and Validation

Team 15

Name	**ID**	**Section**
Abdelrhman Tarek Mohamed	2101547	Sec.1
Ahmed Mahmoud AbdelAzeem	2100718	Sec.1
El mahdy Salih	2101875	Sec.1
Mahmoud Atef	2001313	Sec.1
Zaid Reda Farouk	2101603	Sec.1

Submitted to:

Eng. Abdallah Mohamed

Contents

1 Introduction

The objective of this project ("Part 1") is to build a foundational neural network library to deepen understanding of forward and backward propagation algorithms. The library is designed to be modular, allowing for the construction of sequential models. Key deliverables for this phase include the core library structure, gradient checking validation, and solving the XOR problem as a proof of concept.

2 Library Architecture and Design

The library is structured into modular Python files within the `lib/` directory. The design favors simplicity and clarity while ensuring mathematical correctness.

2.1 Repository Structure

The following directory tree illustrates the organization of the project:

```
.
          .gitignore
          README.md
          requirements.txt
          lib/
                __init__.py
                layers.py        # Dense layer implementation
                activations.py  # ReLU, Sigmoid, Tanh
                losses.py        # MSE Loss
                optimizer.py     # SGD Optimizer
                network.py       # Sequential model class
          notebooks/
                project_demo.ipynb  # Demos: Gradient Check, XOR,
    Autoencoder
          report/
              Part1_Report.pdf
```

2.2 Layer Abstraction (`lib.layers`)

All layers inherit from a base `Layer` class, which defines the interface for `forward` and `backward` passes.

- **Dense Layer**: Implements a fully connected layer.
- **Initialization**: He initialization is used for weights to verify better convergence ($W \sim \mathcal{N}(0, \sqrt{2/n_{in}})$).
- **Gradient Accumulation**: The layer supports gradient accumulation to handle batch processing correctly.

2.3 Activation Functions (`lib.activations`)

Activation functions are implemented as subclasses of `Layer` to integrate seamlessly into the computation graph.

- **ReLU**: $f(x) = \max(0, x)$
- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
- **Tanh**: $f(x) = \tanh(x)$

2.4 Loss Function (`lib.losses`)

The **Mean Squared Error (MSE)** is implemented to measure the discrepancy between predictions and targets.

$$L = \frac{1}{N} \sum (Y_{true} - Y_{pred})^2$$

The class provides both `loss` (forward) and `loss_prime` (gradient w.r.t prediction) methods.

2.5 Optimizer (`lib.optimizer`)

Stochastic Gradient Descent (SGD) is used to update parameters. The `step` method updates weights and biases using the computed gradients and a specified learning rate.

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W}$$

2.6 Network Model (`lib.network`)

The `Sequential` class acts as a container for layers. It manages:

- **Forward Pass**: Sequentially passing input through all layers.
- **Training Loop**: Iterating through epochs, performing forward/backward passes, and invoking the optimizer.

3 Validation and Testing

3.1 Gradient Checking

To ensure the correctness of the analytical backpropagation, we performed a gradient check comparing the analytical gradients against numerical gradients computed via finite differences.

$$\frac{\partial L}{\partial W} \approx \frac{L(W + \epsilon) - L(W - \epsilon)}{2\epsilon}$$

Results:

- **Analytic Gradient**: `[[-2.07469049], [-0.43725958]]`
- **Relative Difference**: 2.81×10^{-12}

The extremely small difference confirms the implementation of backpropagation is correct.

3.2 XOR Problem Validation

The XOR function is a classic non-linear classification problem that requires a multi-layer perceptron to solve.

3.2.1 Network Architecture

A 2-layer network was constructed:

1. **Input Layer**: 2 neurons
2. **Hidden Layer**: Dense(16 units) + Tanh activation
3. **Output Layer**: Dense(1 unit) + Sigmoid activation

3.2.2 Training Parameters

- **Epochs**: 10,000
- **Learning Rate**: 1.0
- **Loss Function**: MSE

3.2.3 Results

The network successfully converged, achieving a final loss of approximately **0.00003**.

Training Log Below is a snippet of the training progress showing the reduction in error over epochs:

```
Training XOR Network
==================================================
Epoch 1/10000       error=0.270882
...
Epoch 1001/10000    error=0.000429
...
Epoch 5001/10000    error=0.000064
...
Epoch 9901/10000    error=0.000030
==================================================
Training Complete
```

Table 1: Final Predictions on XOR Dataset

Input (x1, x2)	Truth (y)	Predicted Probability	Rounded Prediction
(0, 0)	0	0.003440	0
(0, 1)	1	0.994145	1
(1, 0)	1	0.994501	1
(1, 1)	0	0.006622	0

The model achieved **100% accuracy** on the XOR dataset, validating the capability of the library to learn non-linear decision boundaries.

4 Conclusion

The core neural network library has been successfully implemented and verified. The modular design allows for easy extension, and the gradient checking and XOR tests provide strong evidence of its correctness. This foundation is ready for more complex tasks such as the Autoencoder implementation planned for Part 2.