# Neural Network Library - Project Report

## 1. Library Design and Architecture

1. **Library Design and Architecture**

The neural network library was designed to be modular, readable, and easily extensible, similar to simplified versions of PyTorch or Keras. The core idea is to separate every function of a neural network into clearly defined components:

2. **Layer Abstraction (Layer Base Class)**

All layers in the library inherit from a simple Layer base class. This ensures every layer implements:

forward(x) — computes the output of the layer

backward(grad_output) — computes gradients and sends gradient backward

(optional) update(optimizer) — updates weights if the layer has parameters

This abstraction allows stacking any sequence of layers inside a Sequential model.

3. **Dense Layer (Dense)**

The Dense (fully-connected) layer is the main trainable component of the network.

Each Dense layer stores:

Weights W

Biases b

Gradients dW, db

Input cache (needed for backward pass)

Forward pass:

$y=xW+b$

Backward pass computes:

Gradient wrt weights

Gradient wrt biases

Gradient wrt inputs

This structure mirrors how professional deep learning frameworks implement linear layers.

4. **Activation Functions**

Activations are implemented as separate layers (e.g., Tanh, Sigmoid). They do not have weights, only formulas for:

forward activation

backward derivative

This modularity allows replacing activations without modifying the network architecture.

5. **Loss Functions (MSELoss)**

Loss functions are isolated from the model and only compute:

forward(y_true, y_pred)

backward() → gradient wrt predictions

This design keeps the model focused on learning rather than evaluating error.

6. **Optimizer (SGD)**

The SGD optimizer handles weight updates:

$W = W - \eta \cdot dW$

7. **Model Container (Sequential)**

The Sequential class stores a list of layers and manages:

Forward propagation through all layers

Backward propagation from last to first layer

Calling each layer's optimizer update

Running training loops with epochs

It behaves similarly to tf.keras.Sequential.

8. **Directory Structure**

The library is organized inside the AROGO/ folder:

AROGO/ │── layers.py │── activations.py │── losses.py │── optimizer.py └── network.py

This clean separation makes the codebase:

Easier to maintain

Easy to extend

Structured like real machine-learning libraries

Summary

The core principles of the library are:

Modularity (each component is independent)

Clarity (easy to read and understand)

Extensibility (easy to add new layers/activations)

Realistic design inspired by PyTorch/Keras

## 2. Gradient Checking

1. **Analytical Gradients:**
    - Computed using the `backward()` methods of our layers.
2. **Numerical Gradients:**
    - Approximated with the formula:
      [ \frac{\partial L}{\partial W} \approx \frac{L(W + \epsilon) - L(W - \epsilon)}{2 \epsilon} ]
    - Here, ( \epsilon ) is a very small number (default ( 1e-5 )).
3. **Comparison:**
    - We calculate the maximum difference between analytical and numerical gradients.
    - If the difference is very small (e.g., less than (1e-6)), our backpropagation is likely correct.
4. **Results:**

Analytical: [[-0.00232373 -0.00274231 -0.00977339 0.01306674] [-0.00434791 0.01133584 0.002093 0.01870656]] Numerical: [[-0.00232373 -0.00274231 -0.00977339 0.01306674] [-0.00434791 0.01133584 0.002093 0.01870656]] Max Diff: 2.9750407866402373e-12

## 3. XOR Experiment

Epoch 0, Loss: 0.2539001108002241 Epoch 2000, Loss: 0.006986212600061806 Epoch 4000, Loss: 0.002426433168836391 Epoch 6000, Loss: 0.00142500829139407 Epoch 8000, Loss: 0.0009993669268076304 Predictions after training: [[0.01332864] [0.97127211] [0.97049072] [0.03451367]]

## 4. TensorFlow Baseline

1- ease of implementaion : our lib has more ease of implementaion than tensor flow (keras)

2-training time : our lib has less training time than tensor flow (keras)

3-final predections :

our lib results : [[0.01332864] [0.97127211] [0.97049072] [0.03451367]]

tensor flow (keras) : [[0.01457623] [0.9690686 ] [0.97115225] [0.03383316]]