Assignment 3:

Multi-Layer Artificial Neural Network Implementation and Comparison

Submitted by:

Noa Magrisso - 206934978
Shaked Tayouri - 2

1. Introduction

This assignment focuses on implementing and comparing artificial neural networks (ANNs) with one and two hidden layers for classifying handwritten digits from the MNIST dataset. The project extends the implementation from Chapter 11 of *Machine Learning with PyTorch and Scikit-Learn* by Raschka et al. (2022) and evaluates model performance in comparison to a fully connected ANN implemented in Keras/TensorFlow.

2. Implementation Details and Solution Highlights

2.1 Building the Models

2.1.1 One-Hidden-Layer ANN

- Implemented following the **original** Chapter 11 framework.
- Uses a single hidden layer with a **sigmoid** activation function.
- Trained using gradient descent with **mean squared error (MSE) loss**.

2.1.2 Two-Hidden-Layer ANN

- Extended version with **two** fully connected hidden layers.
- Uses **ReLU** activation for improved convergence.
- Added **regularization techniques**, such as **L2 regularization** and **dropout**, to prevent overfitting.

2.1.3 Fully Connected Keras ANN

- Implemented using Keras.
- Uses a **two-hidden-layer** architecture with **ReLU** activation.
- Trained with the **Adam optimizer** and **categorical cross-entropy loss**, leveraging the highly optimized Keras library for better performance.

2.2 Training: Forward and Backward Propagation

2.2.1 Custom Models

In our custom neural network implementation, we explicitly define both **forward propagation** (calculating activations and outputs) and **backward propagation** (computing gradients and updating weights).

Forward Propagation:

- Input features **X** are passed through the layers.
- Each layer applies a **weighted sum** followed by an **activation function**.
- The final layer uses **SoftMax** to compute class probabilities.

Backward Propagation:

- Computes gradients for each layer using the **chain rule**:
    - Output layer gradients are derived from the **loss function** (MSE or cross-entropy).
    - Hidden layer gradients propagate backward using **weight updates**.
- Weight updates are performed using **gradient descent**.

2.2.2 Keras Model:

Unlike the custom model, **Keras abstracts forward and backward propagation**, handling these steps internally.

Forward propagation: Automatically applies matrix multiplications and activations for each layer.

Backward propagation: Uses **automatic differentiation** to compute gradients for weight updates.

Weight updates: Uses **Adam optimizer**, which improves upon standard gradient descent by using momentum and adaptive learning rates.

3. Experimental Setup

- **Dataset:** MNIST (handwritten digits, 10 classes).
- **Data Split:** 70% Training, 30% Testing.
- **Validation:** Different notebooks experiment with various validation set sizes and training parameters.
- **Batch Size:** 100.
- **Epochs:** 50.

4. Evaluation Metrics

- **Accuracy**: Measures the percentage of correct predictions.
- **Macro AUC**: Evaluates the ability of the model to distinguish between classes.
- **MSE (Mean Squared Error)**: Measures the difference between predicted probabilities and true labels.

5. Results & Comparisons

Seed 42, Validation – 10% from training set:

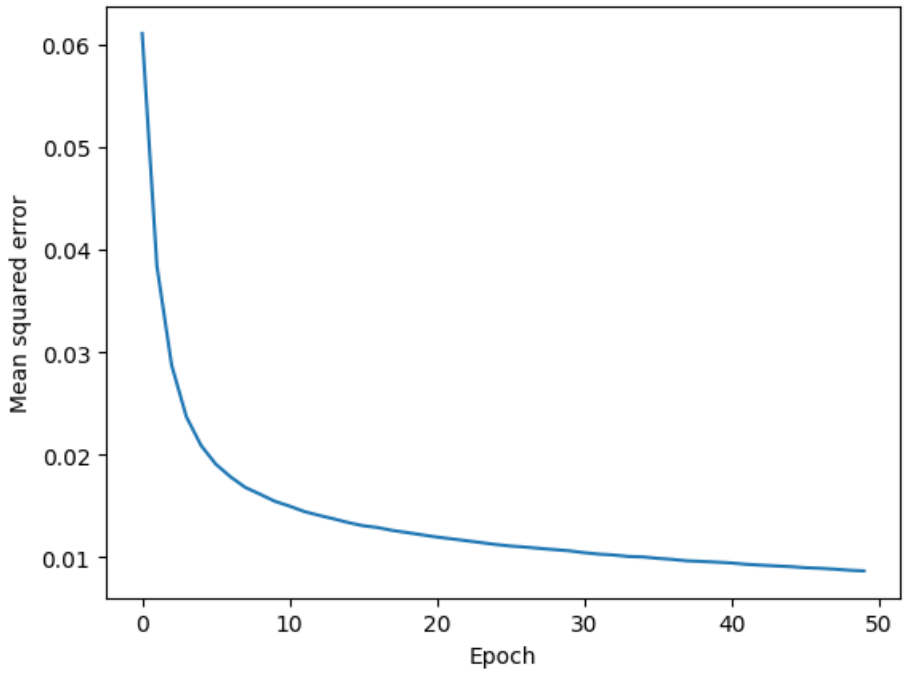
Model	***Accuracy***	***Macro AUC***	***MSE***
One-Hidden-Layer ANN	*94.43%*	*99.18%*	0.0097
Two-Hidden-Layer ANN	*94.55%*	*99.33%*	0.0091
Fully Connected Keras ANN	*96.64%*	*99.89%*	0.0051

Observations:

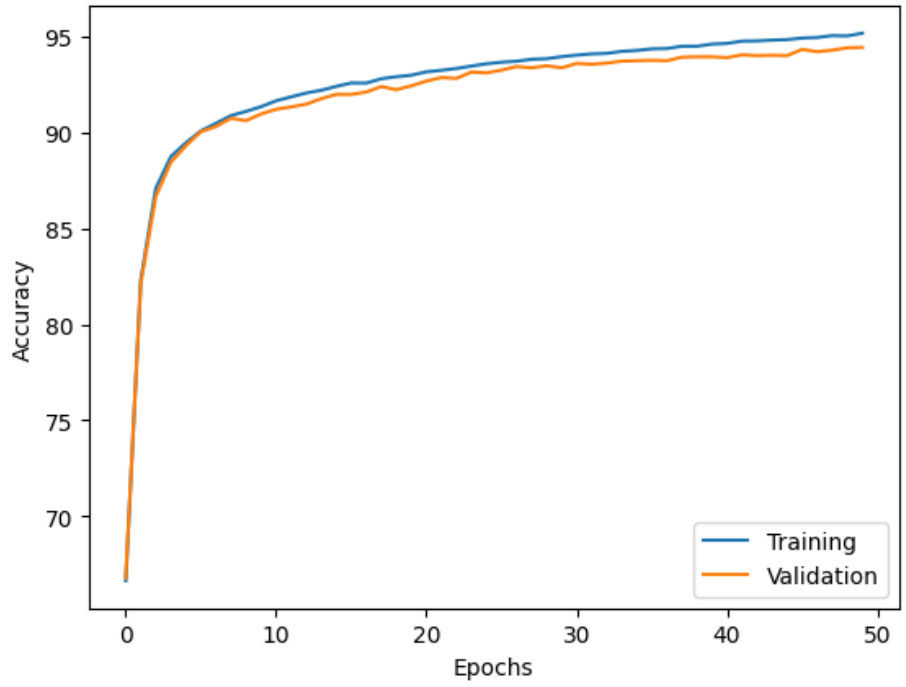
- The two-hidden-layer ANN generally outperformed the single-layer ANN due to its increased model capacity and better activation functions.
- The fully connected Keras ANN benefited from optimized training techniques, leading to better convergence and higher accuracy.
- MSE was lower for models that utilized ReLU activation and proper regularization.

For 1 hidden layer:

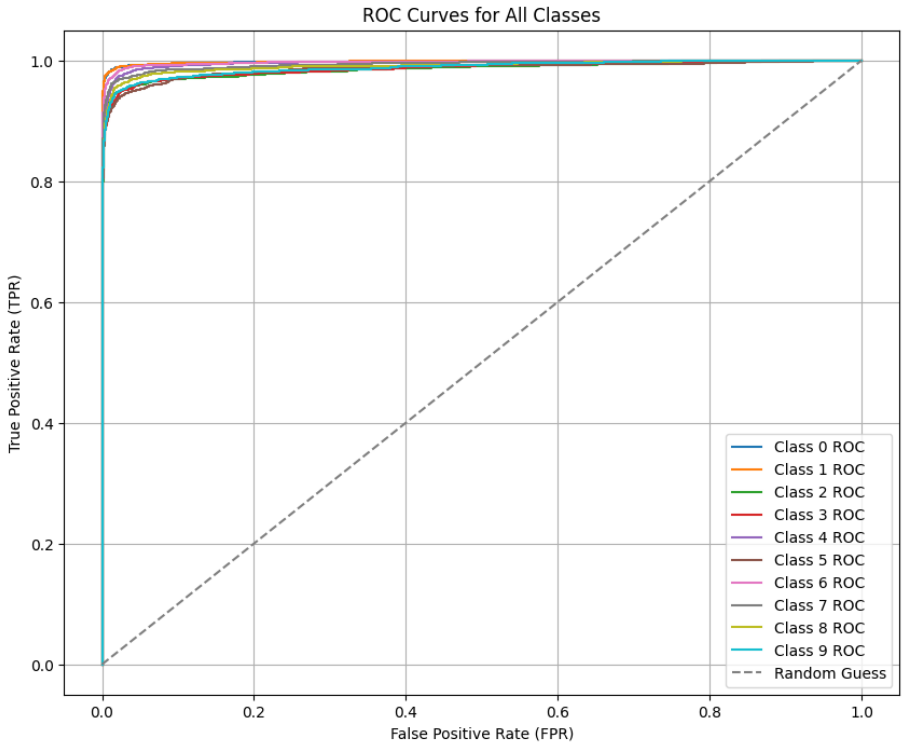
Mean Squared Error Plot



Accuracy Plot

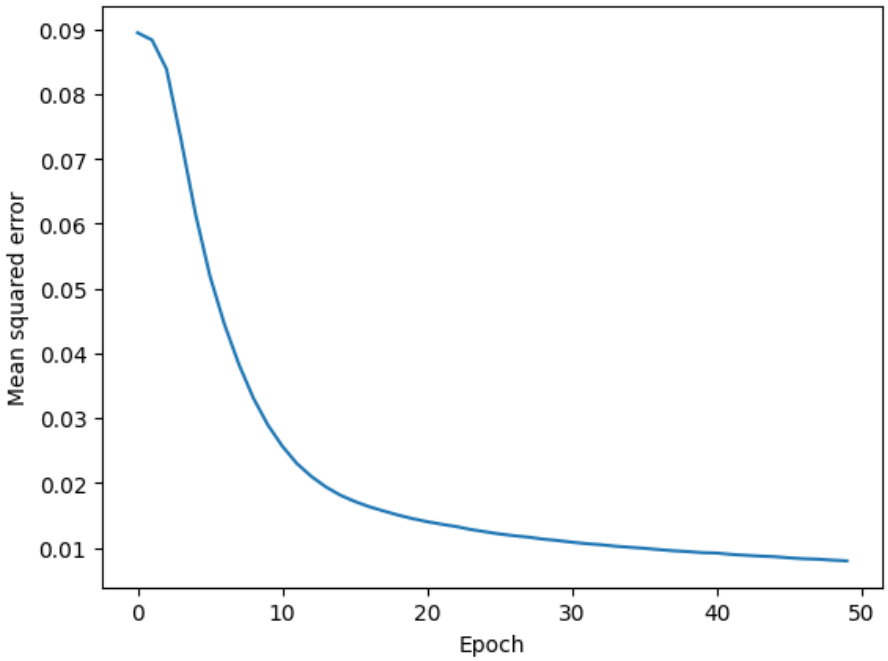


ROC Plot

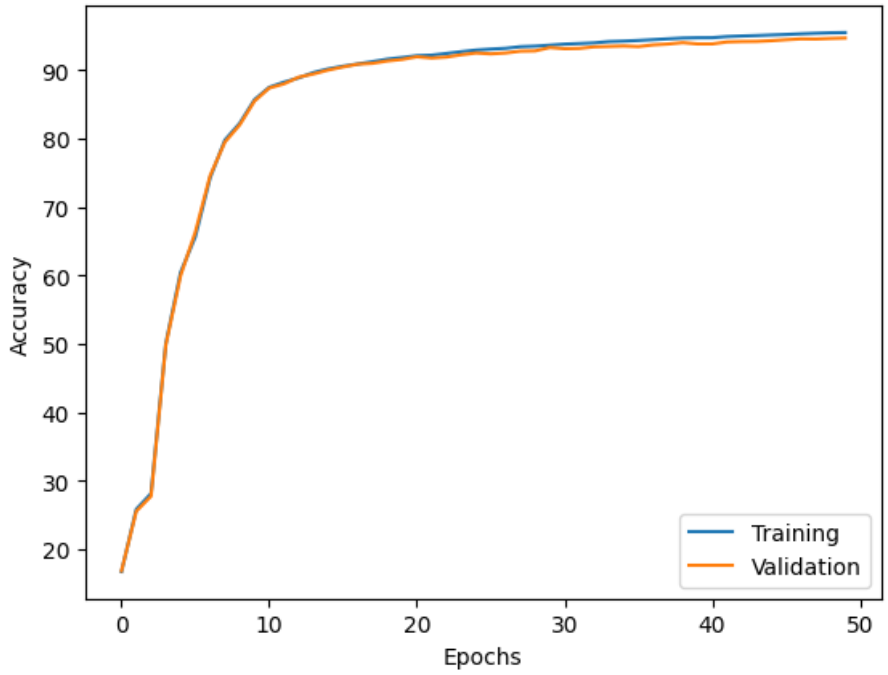

ROC Curves for All Classes
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 ROC
Class 1 ROC
Class 2 ROC
Class 3 ROC
Class 4 ROC
Class 5 ROC
Class 6 ROC
Class 7 ROC
Class 8 ROC
Class 9 ROC
Random Guess

For 2 hidden layers:

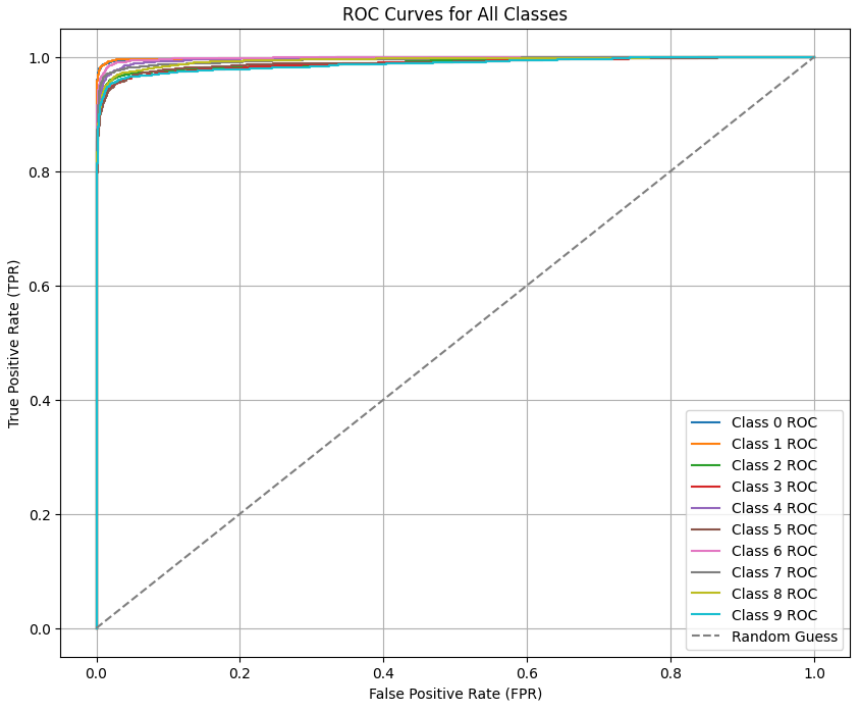
Mean Squared Error Plot



Accuracy Plot

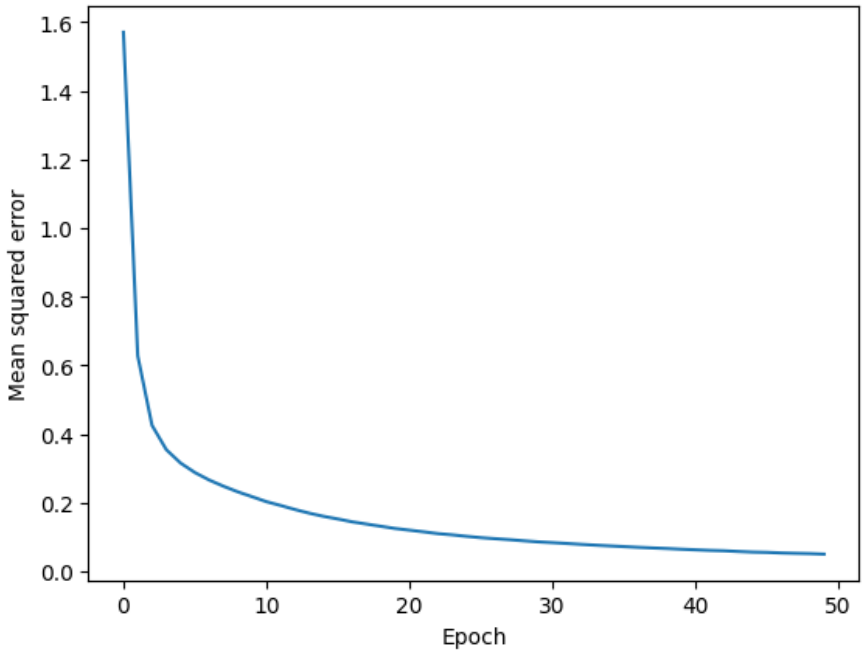


ROC Plot

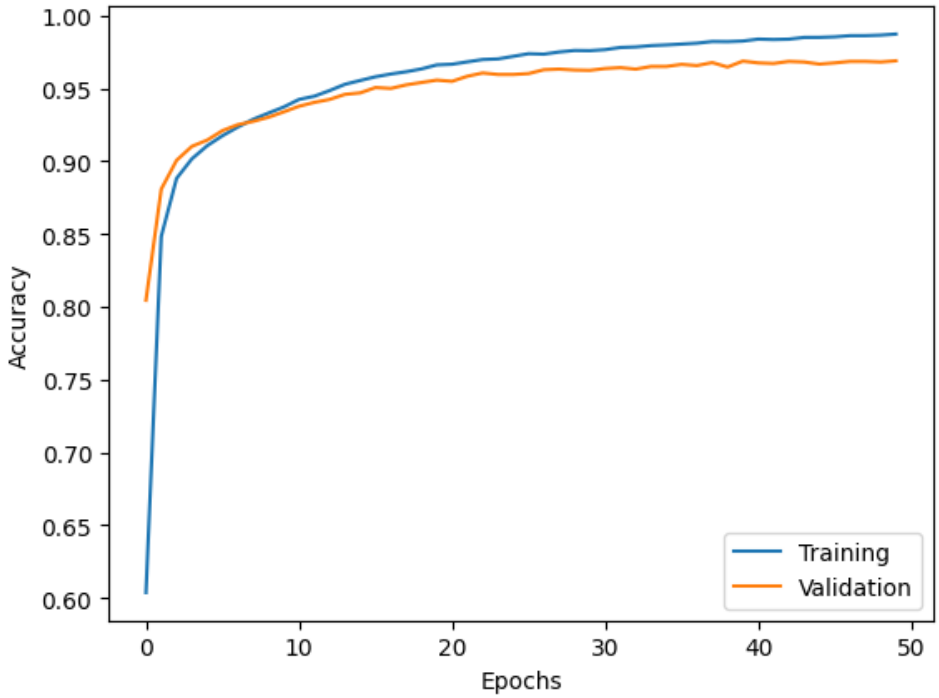

ROC Curves for All Classes
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 ROC
Class 1 ROC
Class 2 ROC
Class 3 ROC
Class 4 ROC
Class 5 ROC
Class 6 ROC
Class 7 ROC
Class 8 ROC
Class 9 ROC
Random Guess

For fully connected ANN by Keras implementation:

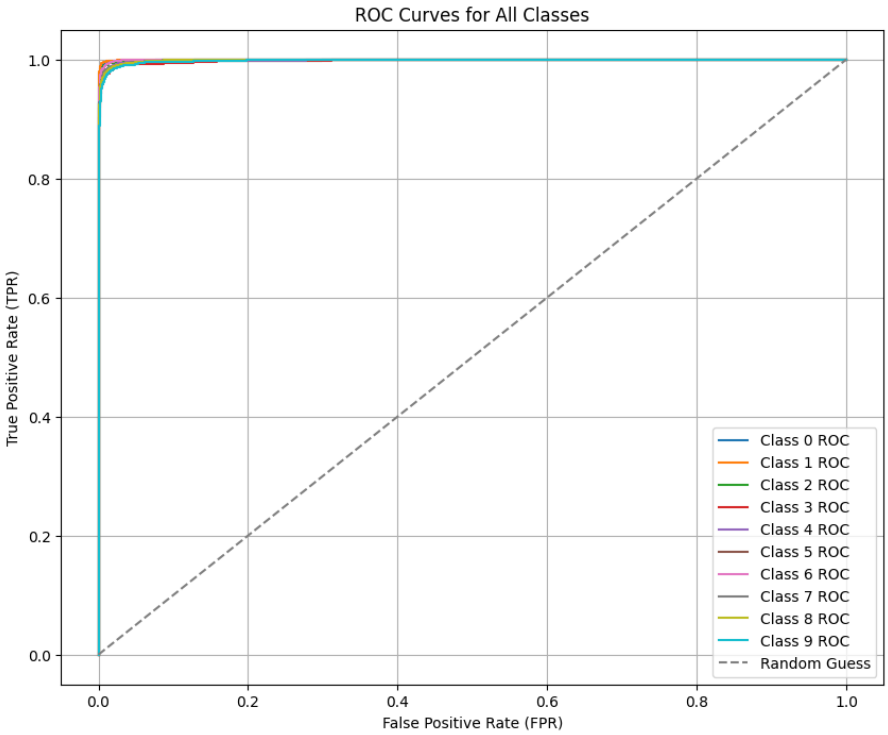
Mean Squared Error Plot



Accuracy Plot



ROC Plot


ROC Curves for All Classes
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 ROC
Class 1 ROC
Class 2 ROC
Class 3 ROC
Class 4 ROC
Class 5 ROC
Class 6 ROC
Class 7 ROC
Class 8 ROC
Class 9 ROC
Random Guess

6. Conclusion

- The two-hidden-layer ANN showed improved performance over the one-hidden-layer model, validating the effectiveness of deeper networks for MNIST classification.
- The Keras ANN provided an additional benchmark, demonstrating the advantages of pre-optimized deep learning libraries.
- Further improvements could be achieved by tuning hyperparameters, using convolutional networks, or experimenting with advanced optimizers.

7. References

1. **Machine Learning with PyTorch and Scikit-Learn**:
    - Raschka, S., Liu, Y., & Mirjalili, V. (2022). "Implementing a Multi-layer Artificial Neural Network from Scratch."
    - Chapter 11 on GitHub
2. **MNIST Dataset**:
    - Yann LeCun and Corinna Cortes. "The MNIST Database of Handwritten Digits."
    - Website Link
3. **Keras Documentation**:
    - Keras: "Building Fully Connected Neural Networks."
    - Keras Documentation

The full implementation can be found in the GitHub repository.