Assignment 1: Forward and Backward Propagation

Yassine Ibork
Y723u998
Deep Learning CS-898BD

Supervised by: Dr. Lokesh Das
September 2024

I. Introduction

This assignment is designed to provide a hands-on understanding of deep learning by first exploring the mechanisms of forward and backward propagation. This was accomplished through the initial phase, where we performed both a forward and a backward pass on a simple function. Furthermore, we developed our technical skills in deep learning by experimenting with various models featuring different architectures. These models were trained using the MNIST dataset, which comprises images of handwritten digits from 0 to 9. My code will be found under this github repository: https://github.com/yibork/assignment1

II. Methodology

Since this assignment is divided into two parts, I will start by discussing the methodology followed in the first part, then proceed to the second part.

Part I: Forward and Backward Propagation

This part focuses on the calculations of forward and backward propagation for the function:

1. Drawing a Computational Graph

The function is given by:

$$f(x) = (x_1 w_1 + x_2 w_2) \cdot (x_3 w_3 + x_4 w_4)$$

2. Forward Propagation

To perform forward propagation, the following steps are carried out:

a. $q = x_1 \cdot w_1 = 0.7 \cdot (-1.5) = -1.05$

b. $t = x_2 \cdot w_2 = 0.34 \cdot (-0.35) = -0.119$

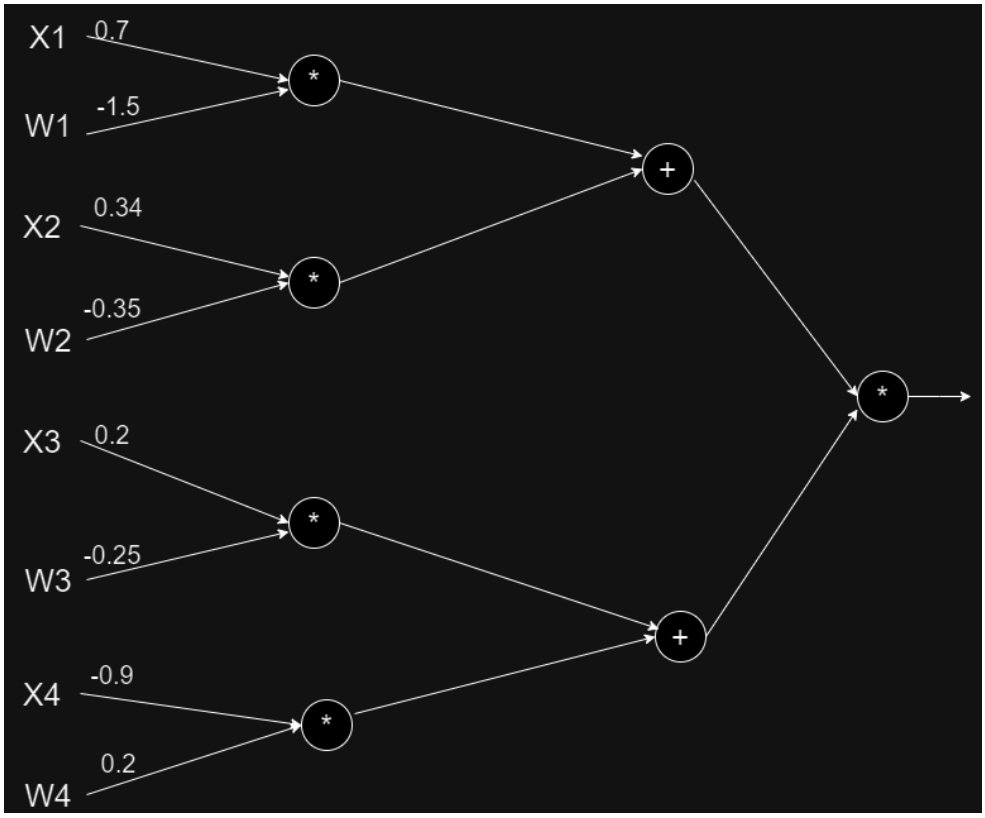

Figure 1: Calculation of q and t

c. $s = x_3 \cdot w_3 = 0.2 \cdot (-0.25) = -0.05$

d. $g = x_4 \cdot w_4 = (-0.9) \cdot 0.2 = -0.18$

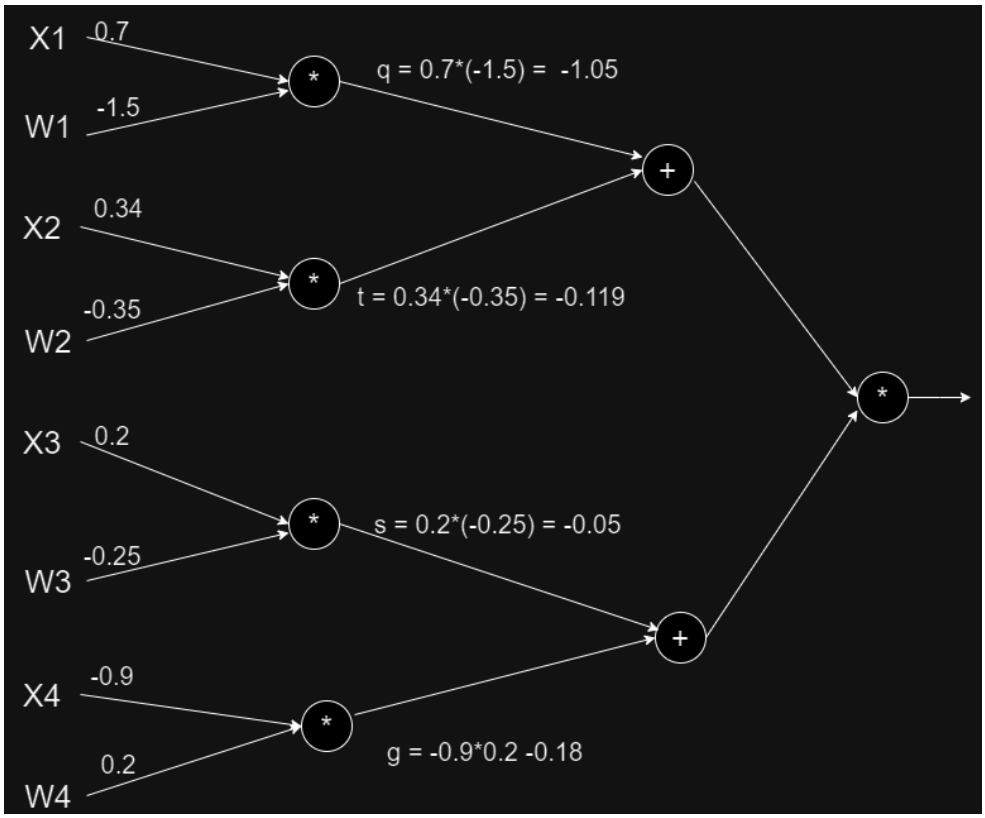

Figure 2: Calculation of s and g

e. $p = q + t = (-1.05) + (-0.119) = -1.169$

f. $r = s + g = (-0.05) + (-0.18) = -0.23$

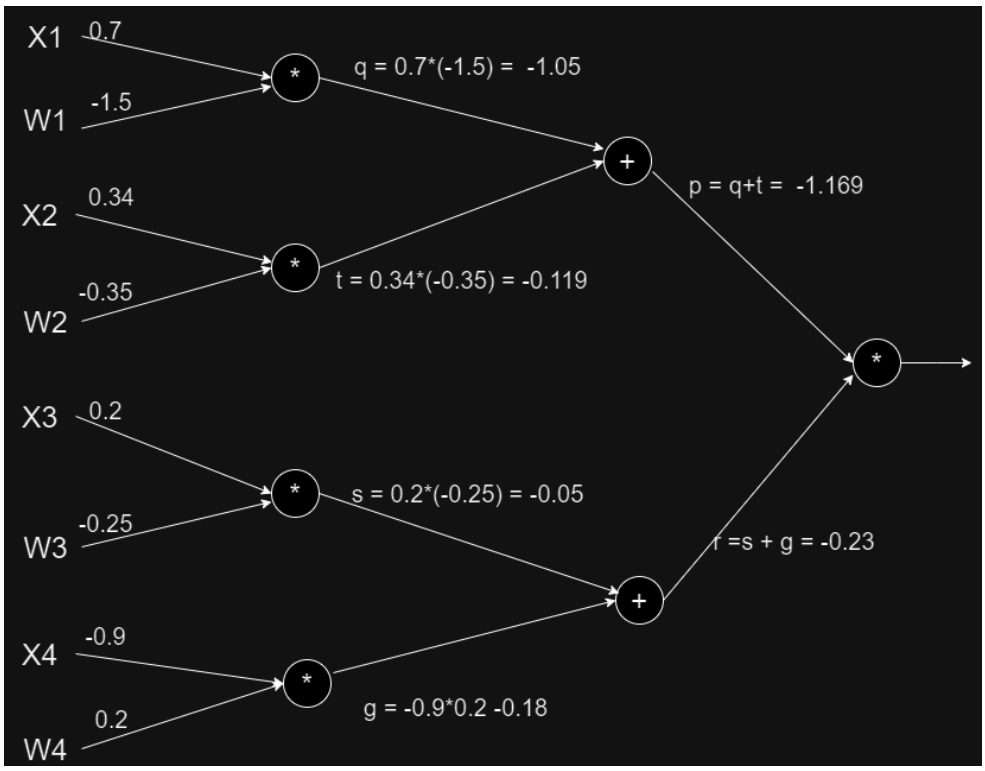

Figure 3: Calculation of p and r

g. $f = p \cdot r = (-1.169) \cdot (-0.23) = 0.26887$

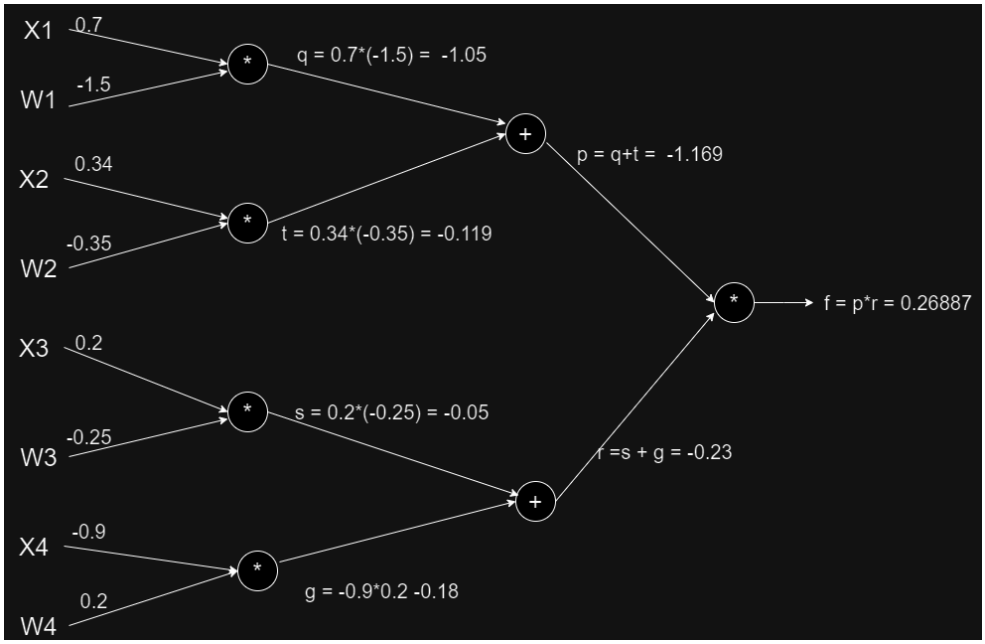

Figure 4: Final calculation of f

3. Backward Propagation

The following partial derivatives are computed:

$$q = x_1 \cdot w_1, \quad \frac{\partial q}{\partial x_1} = w_1, \quad \frac{\partial q}{\partial w_1} = x_1$$

$$t = x_2 \cdot w_2, \quad \frac{\partial t}{\partial x_2} = w_2, \quad \frac{\partial t}{\partial w_2} = x_2$$

$$s = x_3 \cdot w_3, \quad \frac{\partial s}{\partial x_3} = w_3, \quad \frac{\partial s}{\partial w_3} = x_3$$

$$g = x_4 \cdot w_4, \quad \frac{\partial g}{\partial x_4} = w_4, \quad \frac{\partial g}{\partial w_4} = x_4$$

$$p = q + t, \quad \frac{\partial p}{\partial q} = 1, \quad \frac{\partial p}{\partial t} = 1$$

$$r = s + g, \quad \frac{\partial r}{\partial s} = 1, \quad \frac{\partial r}{\partial g} = 1$$

$$f = p \cdot r, \quad \frac{\partial f}{\partial p} = r, \quad \frac{\partial f}{\partial r} = p$$

We want to find:

$$\frac{\partial f}{\partial p} = r = -0.23$$

$$\frac{\partial f}{\partial r} = p = -1.169$$

Now, computing the gradients:

Gradients with respect to x_1 and w_1:

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial x_1}$$

$$\frac{\partial f}{\partial x_1} = \left(\frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial q} \cdot \frac{\partial q}{\partial x_1} \right) = r \cdot 1 \cdot w_1 = -0.23 \cdot (-1.5) = 0.345$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial w_1}$$

$$\frac{\partial f}{\partial w_1} = \left(\frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial q} \cdot \frac{\partial q}{\partial w_1} \right) = r \cdot 1 \cdot x_1 = -0.23 \cdot 0.7 = -0.161$$

Gradients with respect to x_2 and w_2:

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial x_2}$$

$$\frac{\partial f}{\partial x_2} = \left(\frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial t} \cdot \frac{\partial t}{\partial x_2} \right) = -0.23 \cdot 1 \cdot w_2 = -0.23 \cdot (-0.35) = 0.0805$$

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial w_2}$$

$$\frac{\partial f}{\partial w_2} = \left(\frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial t} \cdot \frac{\partial t}{\partial w_2} \right) = -0.23 \cdot 1 \cdot x_2 = -0.23 \cdot 0.34 = -0.0782$$

Gradients with respect to x_3 and w_3:

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial x_3}$$

$$\frac{\partial f}{\partial x_3} = \left(\frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial s} \cdot \frac{\partial s}{\partial x_3} \right) = -1.169 \cdot 1 \cdot w_3 = -1.169 \cdot (-0.25) = 0.29225$$

$$\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial w_3}$$

$$\frac{\partial f}{\partial w_3} = \left(\frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial s} \cdot \frac{\partial s}{\partial w_3} \right) = -1.169 \cdot 1 \cdot x_3 = -1.169 \cdot 0.2 = -0.2338$$

Gradients with respect to x_4 and w_4:

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial x_4}$$

$$\frac{\partial f}{\partial x_4} = \left(\frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial g} \cdot \frac{\partial g}{\partial x_4} \right) = -1.169 \cdot 1 \cdot w_4 = -1.169 \cdot 0.2 = -0.2338$$

$$\frac{\partial f}{\partial w_4} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial w_4}$$

$$\frac{\partial f}{\partial w_4} = \left(\frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial g} \cdot \frac{\partial g}{\partial w_4} \right) = -1.169 \cdot 1 \cdot x_4 = -1.169 \cdot (-0.9) = 1.0521$$

detailed steps concerning how I did the calculations are found here

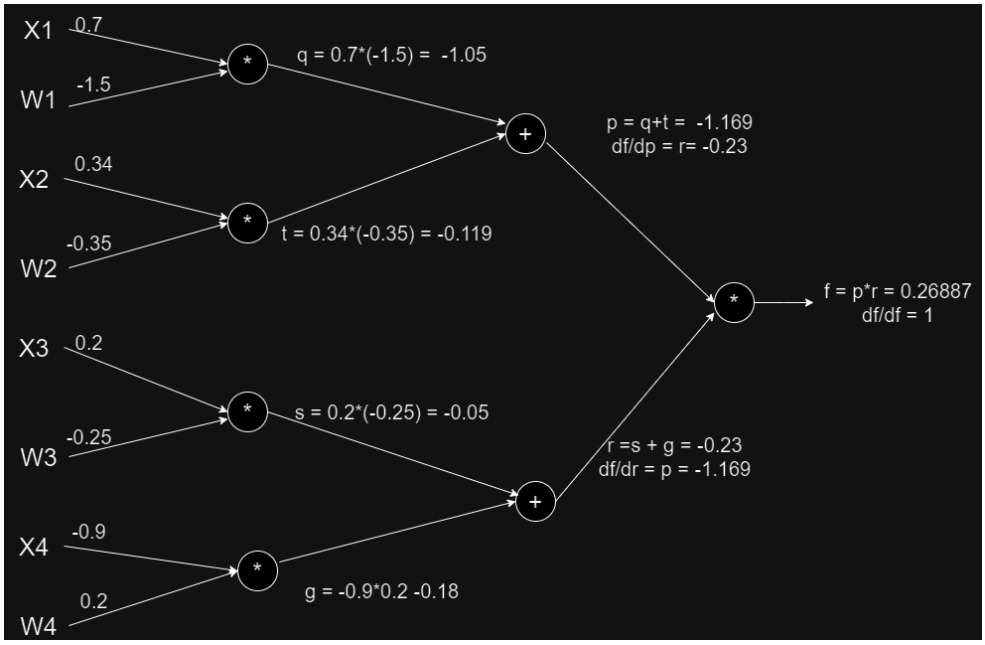

Figure 5: Calculate df/dp and df/dr

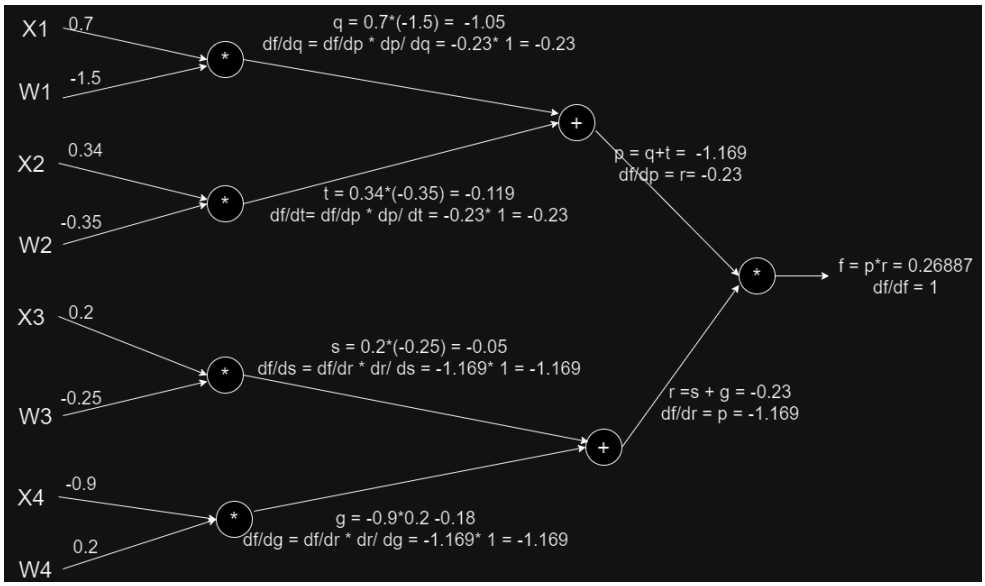

Figure 6: Calculate df/dq, df/dt, df/ds,and df/dg

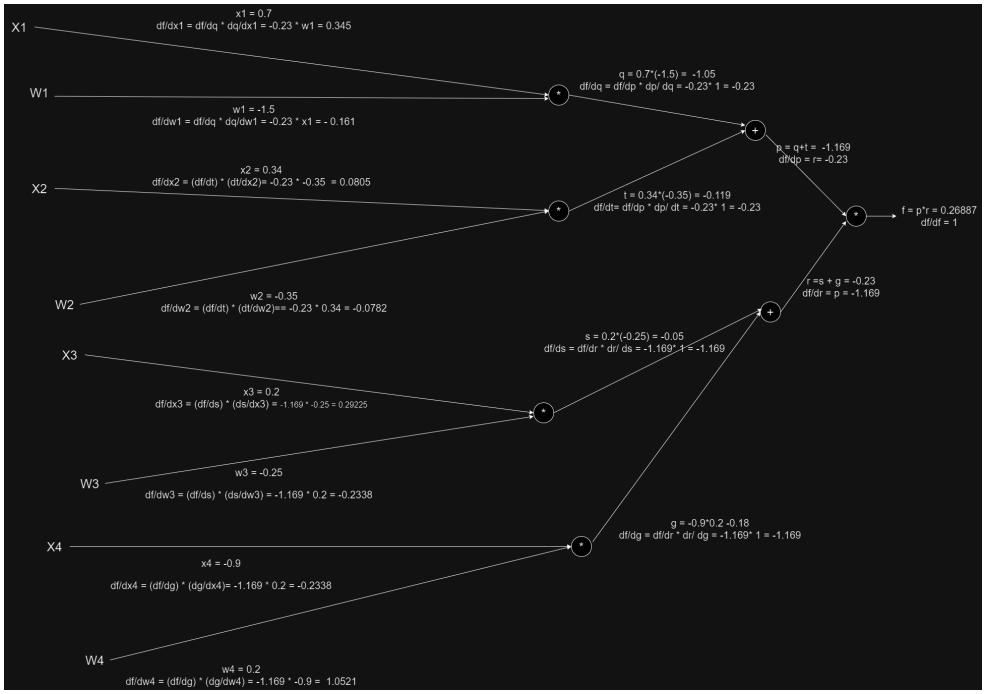

Figure 7: Calculate df/dx1, df/dw1, df/dx2, df/dw2, df/dx3, df/dw3, df/dx4, df/dw4

Final Gradients

$$\frac{\partial f}{\partial x_1} = 0.345, \quad \frac{\partial f}{\partial x_2} = 0.0805, \quad \frac{\partial f}{\partial x_3} = 0.29225, \quad \frac{\partial f}{\partial x_4} = -0.2338$$

$$\frac{\partial f}{\partial w_1} = -0.161, \quad \frac{\partial f}{\partial w_2} = -0.0782, \quad \frac{\partial f}{\partial w_3} = -0.2338, \quad \frac{\partial f}{\partial w_4} = 1.0521$$

Part II: Deep Learning models implementation

For this project, I implemented two distinct deep learning models, referred to as **Network A** and **Network B**. The primary objective of these models was to compare different architectures and training techniques in order to optimize performance on the MNIST dataset, while effectively addressing the issue of overfitting.

Both models were trained using the MNIST dataset, which consists of 60,000 training images and 10,000 validation images. To ensure consistent input to the models, the images were normalized to have a mean of 0.1307 and a standard deviation of 0.3081. This normalization step helps scale the pixel values, which range from 0 to 1, into a more standardized format, improving the models' convergence during training.

Additionally, to create a more rigorous evaluation process, I partitioned 5% of the training data (3,000 images) to form a separate test set. This allowed for better evaluation of the model's generalization ability by testing on data the models had never encountered before. The dataset was divided as follows:

- **Training set size**: 57,000 images
- **Validation set size**: 10,000 images
- **Test set size**: 3,000 images

III. Deep Learning Architecture

For this assignment, I developed two distinct models with different architectures: **Network A** and **Network B**. These architectures vary in the number of layers and the number of neurons per layer,

providing a comparison of their respective performance in terms of classification accuracy and generalization on the MNIST dataset.

Network A Architecture

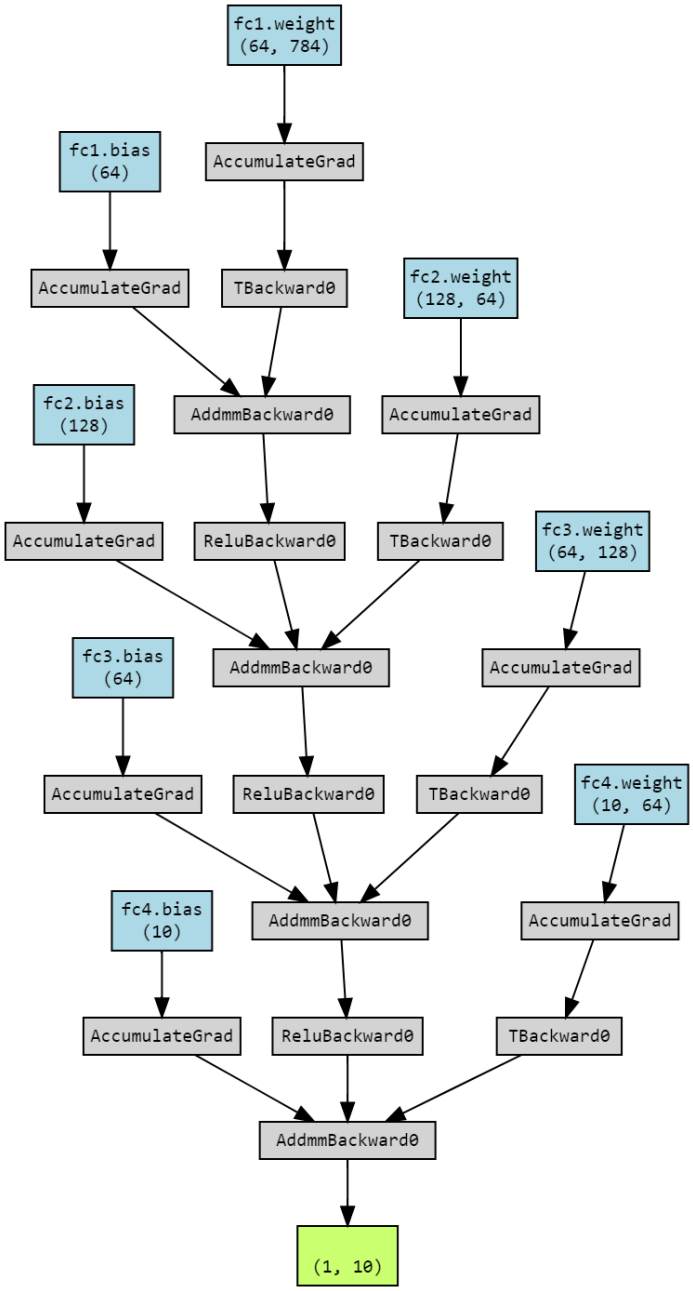

Figure 8: Network A architecture

Network A consists of four fully connected (dense) layers. The input layer flattens the 28x28 pixel images into a vector of 784 features. The network then passes this input through the following layers:

- **First hidden layer**: 64 neurons with a ReLU activation function.
- **Second hidden layer**: 128 neurons with ReLU activation.
- **Third hidden layer**: 64 neurons with ReLU activation.
- **Output layer**: 10 neurons (one for each digit class in the MNIST dataset), producing the final classification output.

The ReLU activation function is used in all hidden layers to introduce non-linearity, while the final output is passed through the softmax function to predict the probabilities of each class. This architecture is relatively simple, designed to balance between model complexity and computational efficiency.

Network B Architecture

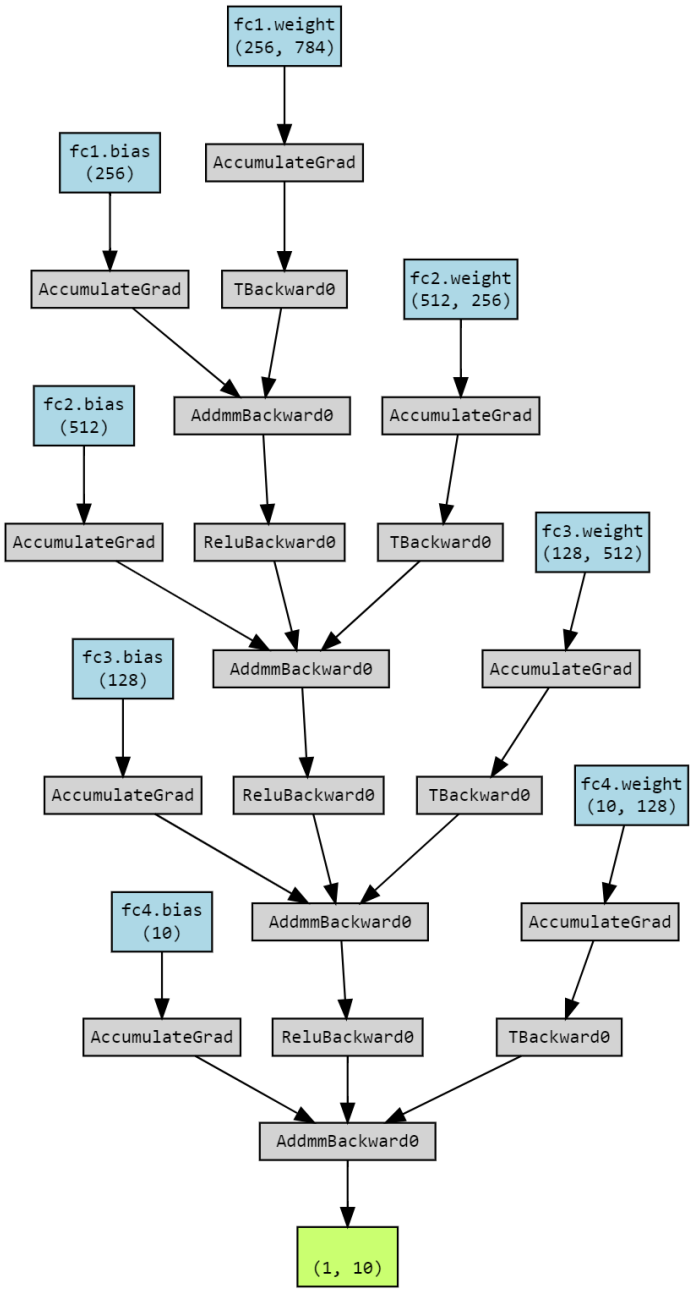

Figure 9: Network B architecture

Network B follows a similar structure but is more complex, with a greater number of neurons in each layer:

- **First hidden layer**: 256 neurons with ReLU activation.
- **Second hidden layer**: 512 neurons with ReLU activation.
- **Third hidden layer**: 128 neurons with ReLU activation.
- **Output layer**: 10 neurons, corresponding to the 10 digit classes in the MNIST dataset.

As in Network A, the ReLU activation function is applied in all hidden layers, and the softmax function is used in the output layer. The increase in the number of neurons in Network B's layers makes it a deeper and more complex model, which can potentially capture more intricate patterns in the data.

IV. Experiments and Results

In this section, I present the results from training two different network architectures, **Network A** and **Network B**, with a focus on their loss trends, error rates, and performance metrics. The results of each experiment are visualized in the following figures.

Hyperparameters and Model Setup

For both **Network A** and **Network B**, I initially used the following hyperparameters during training:

- **Optimizer**: Adam optimizer with a learning rate of 0.001.
- **Loss Function**: Cross-Entropy Loss.
- **Batch Size**: 64.
- **Epochs**: 500.

These hyperparameters were selected to ensure stable learning and convergence during training.

Network A

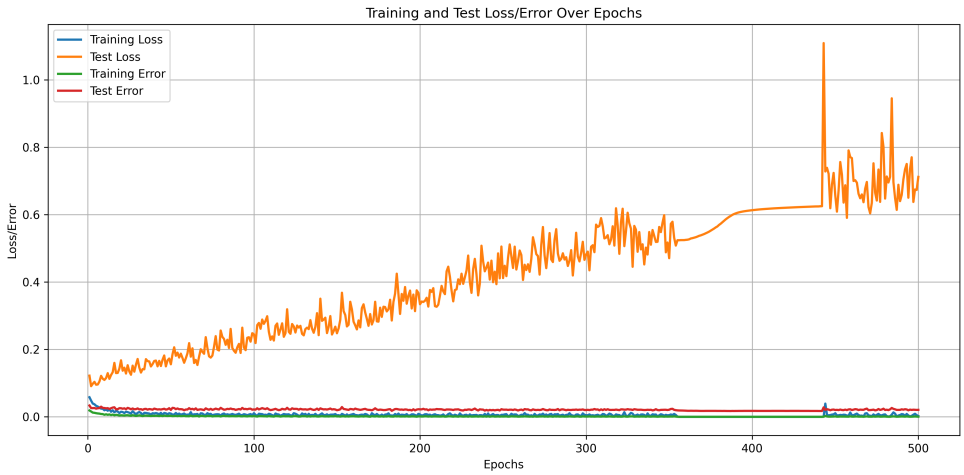

Figure 10: Training and validation loss/error trends for Network A.

The plot in Figure 10 shows the training and validation loss curves, as well as the error rates for Network A over time. The training loss decreased steadily, while the validation loss increased after each epoch, indicating overfitting. To address this issue, I plan to introduce a dropout layer, modify the learning rate, and add a learning rate scheduler.

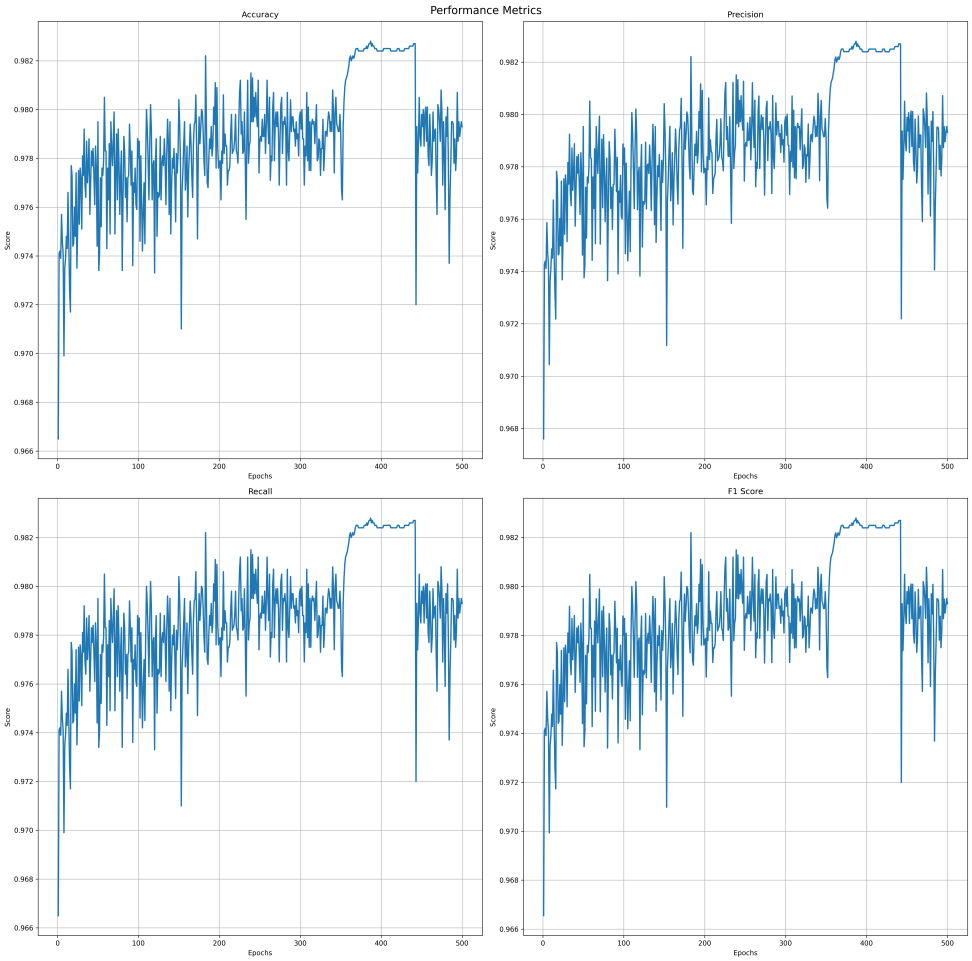

Figure 11: Performance metrics for Network A: accuracy, precision, recall, and F1 score.

Figure 11 illustrates the performance metrics for Network A across multiple epochs. The accuracy, precision, recall, and F1 score follow similar trends. This can be explained by the fact that these metrics are highly correlated in classification tasks, where improving one metric often leads to improvements in the others. By the end of 500 epochs, the model achieved an accuracy, recall, precision, and F1 score of 97.86%. The highest accuracy, 98.22%, was reached at epoch 430. On the test set, which consists of 3,000 images, the model achieved an accuracy of 97.67%.

Network B

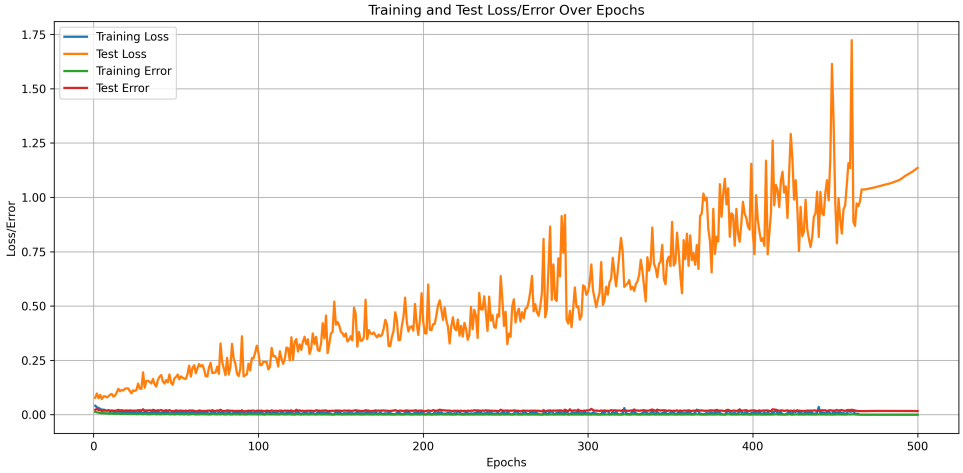

Figure 12: Training and validation loss/error trends for Network B.

After training Network B with the same hyperparameters as Network A, Figure 12 shows the loss and error rate trends for Network B. Similar to Network A, the training loss and error decreased after each epoch, while the validation loss increased, indicating overfitting. Network B also exhibited a higher training loss compared to Network A.

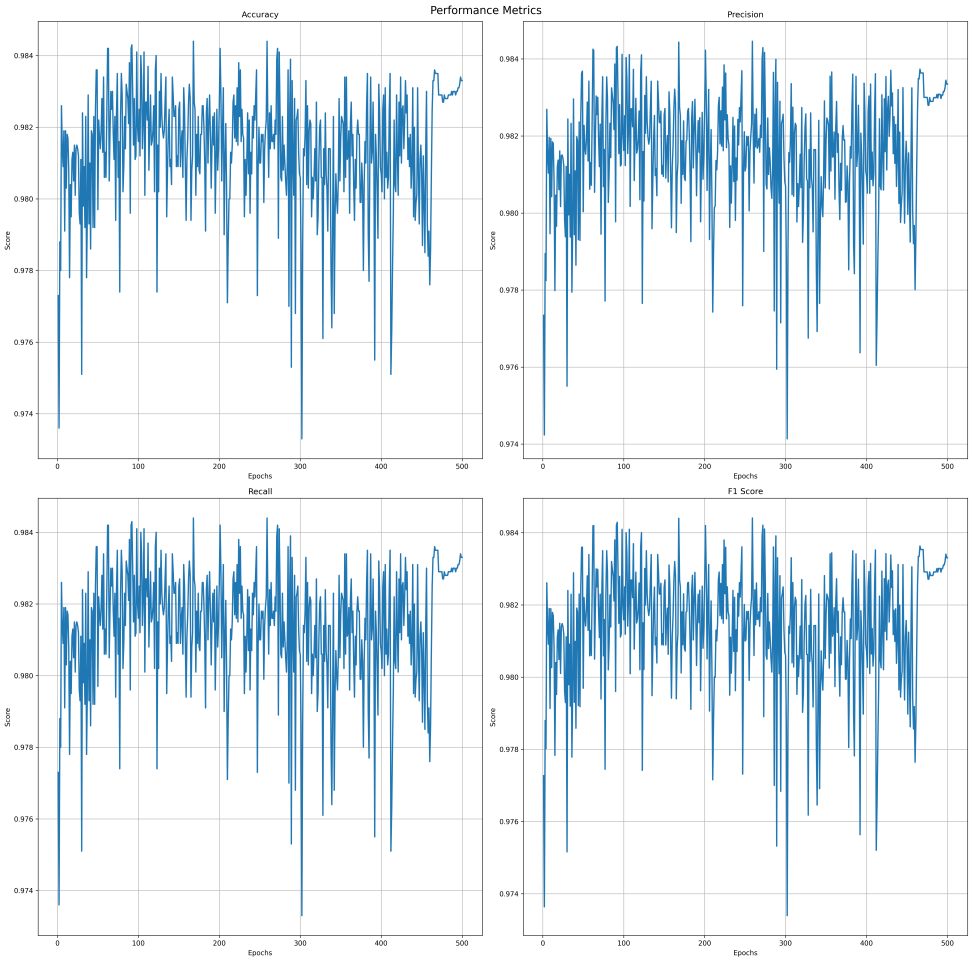

Figure 13: Performance metrics for Network B: accuracy, precision, recall, and F1 score.

The performance metrics for Network B, displayed in Figure 13, demonstrate its classification capabilities. The model achieved a training accuracy, precision, recall, and F1 score of 98.3% by the end of training. The highest accuracy, 98.43%, was reached at epoch 264. A comparison with Network A (Figure 11) reveals that Network B achieved a slightly higher accuracy but exhibited more severe overfitting. On the test set, Network B achieved an accuracy of 98.13%.

Addressing Overfitting

To address the overfitting observed during training, I made the following adjustments to the hyperparameters:

- **Reduced the learning rate**: The initial learning rate of 0.01 was decreased using a learning rate scheduler (StepLR), reducing the rate by a factor of 0.1 every 10 epochs to allow for finer weight updates as training progressed.
- **Applied a dropout of 0.3**: Adding a dropout layer helped reduce the complexity of the model, preventing it from overfitting the training data.

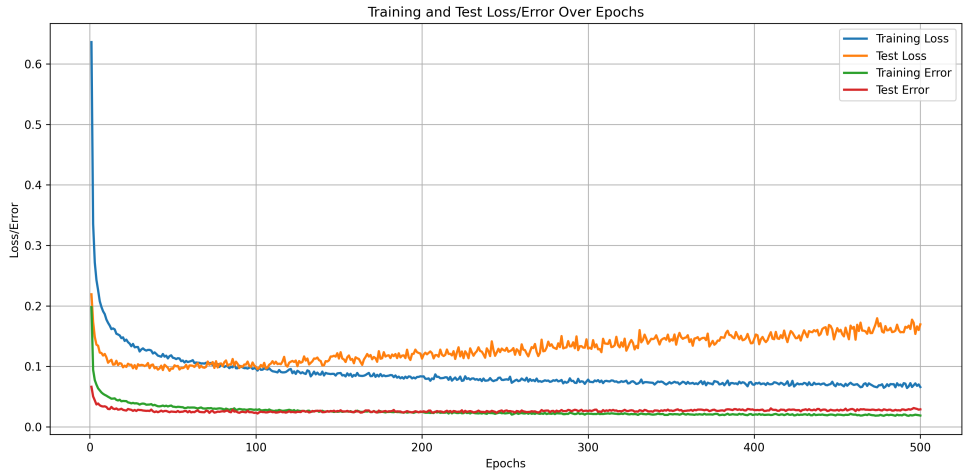

Figure 14: Experiment 2: Training and validation loss for Network A.

After adding dropout to overcome overfitting, we observe a significant reduction in the testing loss from 0.65 to 0.18, as shown in Figure 14. However, the training loss increased slightly, reaching 0.07. This is expected, as dropout increases regularization, which can reduce performance on training data while improving generalization on unseen data.

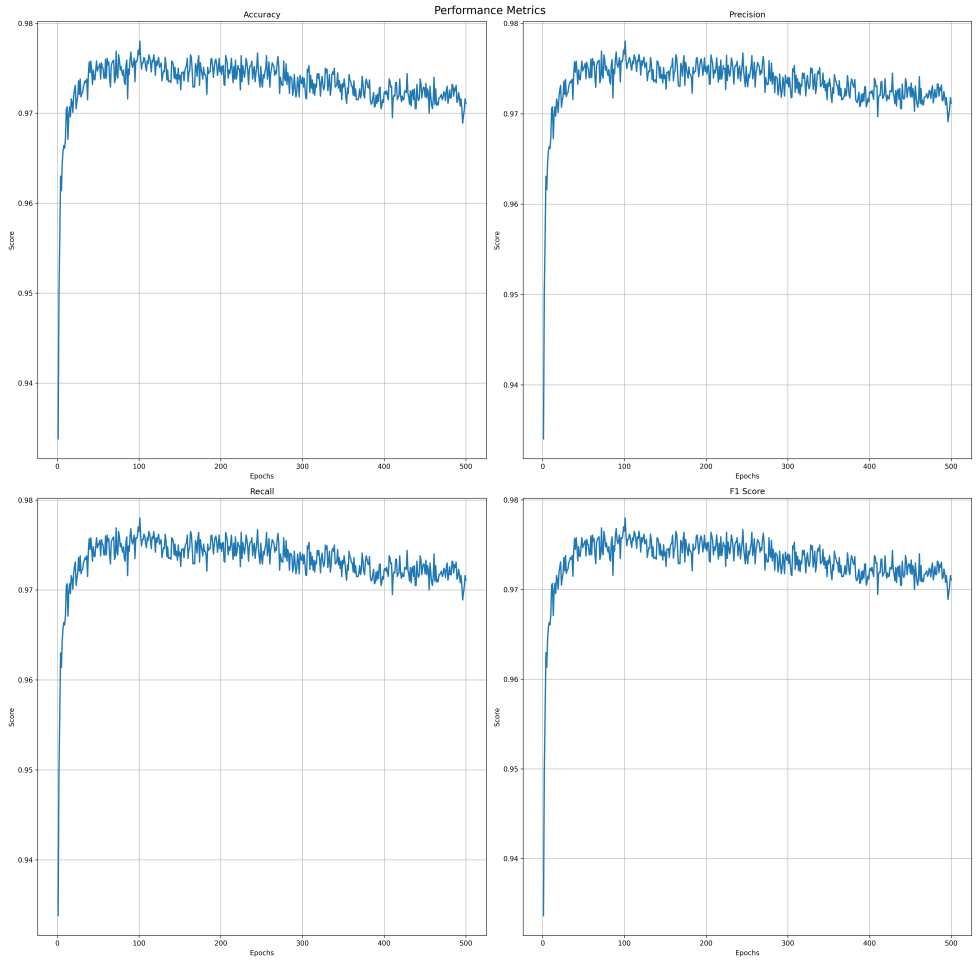

Figure 15: Experiment 2: Performance metrics for Network A: accuracy, precision, recall, and F1 score.

In this setup, the accuracy, precision, recall, and F1 score decreased slightly, reaching 97.2%, compared to the 97.86% achieved in Experiment 1 for Network A. This suggests that the regularization techniques helped reduce overfitting at the cost of a small decrease in performance.

Network B - Experiment 2

Now, I applied the same experiment to Network B.

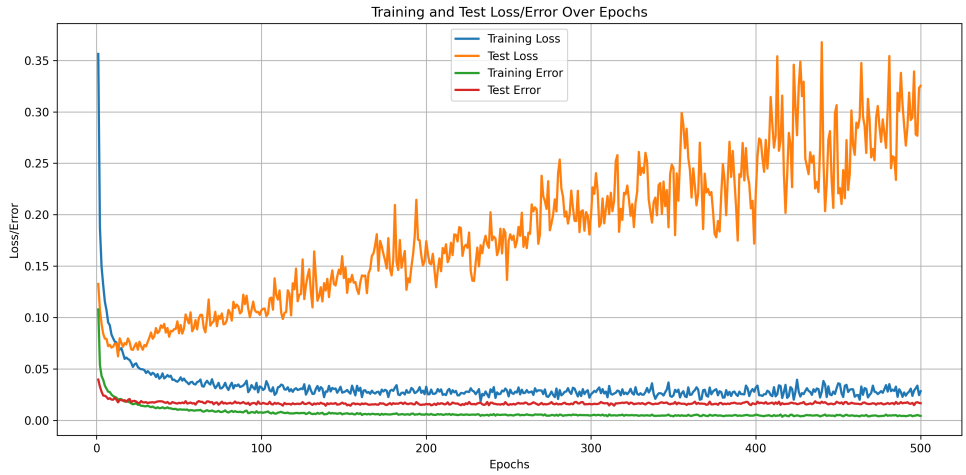

Figure 16: Experiment 2: Training and validation loss for Network B.

In the second experiment on Network B, the testing loss dropped from 1.2 to 0.35, indicating improvement. However, some overfitting remains, likely due to the model's increased capacity or insufficient regularization. Further adjustments, such as increasing dropout or applying stronger regularization, may be necessary.

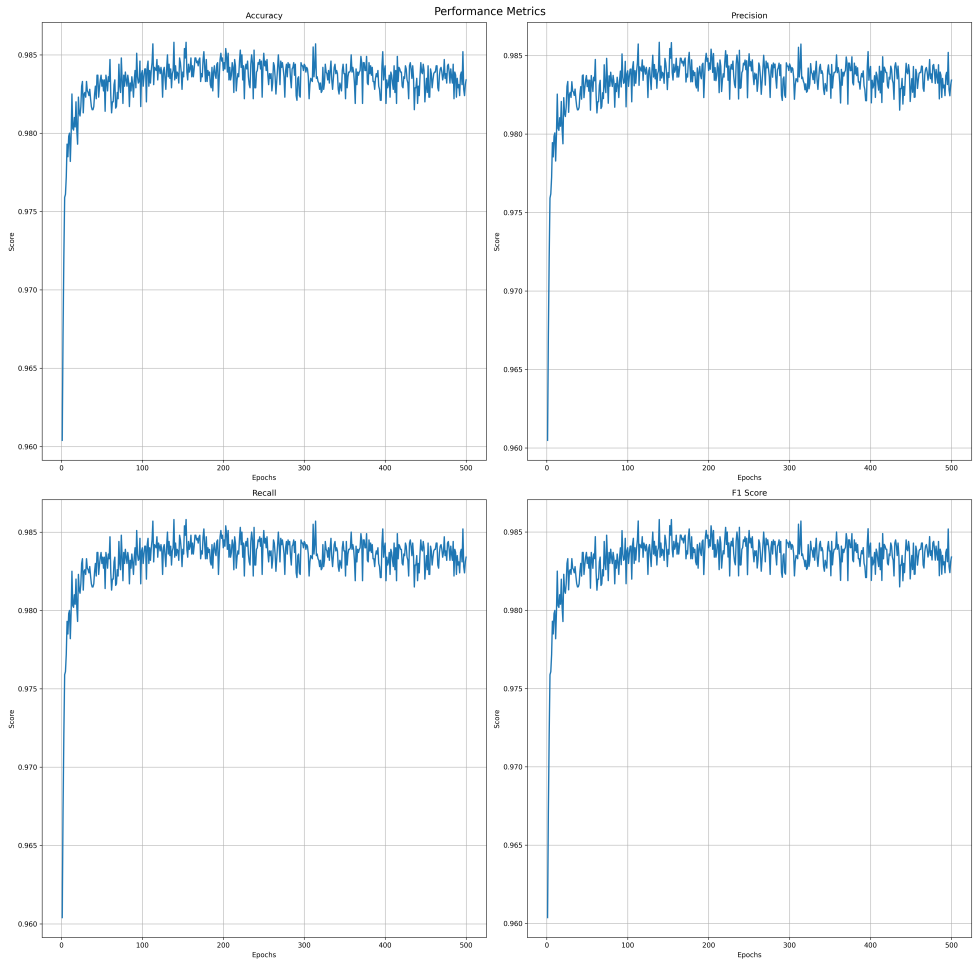

Figure 17: Experiment 2: Performance metrics for Network B: accuracy, precision, recall, and F1 score.

With this second experiment, Network B achieved an accuracy, precision, recall, and F1 score of

98.4%. This slight improvement in metrics suggests that the model benefited from the adjustments, though the presence of some overfitting indicates room for further optimization.

V. Conclusion

This assignment deepened my understanding of forward and backward propagation, as well as the practical skills necessary for designing, training, and evaluating neural networks using the MNIST dataset. By experimenting with different architectures, I gained valuable insights into how network complexity, learning rates, and overfitting can influence model performance. Overall, this project reinforced the critical connection between theory and implementation in deep learning.

VI. References

To implement model training, I referred to several key resources:

- **PyTorch Optimization Documentation**: Guidance on optimization algorithms. https://pytorch.org/
- **PyTorch MNIST Dataset Documentation**: Information on loading and preprocessing the MNIST dataset. https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html
- **PyTorch Tensor Backward Documentation**: Details on gradient computation. https://pytorch.org/
- **Medium Blog on PyTorch Hooks**: Insights on using hooks for debugging and model introspection. https://medium.com/analytics-vidhya/pytorch-hooks-5909c7636fb