Name: Pravallika Cheekatimalla

Assignment: Deep learning Assignment 1

Git Hub: https://github.com/Pravallika-Cheekatimalla/DeepLearning/tree/main/AssignmentOne

Deep vs Shallow

Simulate a function

Settings

- **Task**: is to evaluate the performance of three different neural network architectures in approximating two mathematical functions.
- **Function 1**: $y=\sin(5\pi x)/ 5\pi x$
- **Function 2**: $y=\text{sign}(\sin(5\pi x))$
- **Learning Rate**: 0.001
- **Optimizer**: Adam
- **Model Architectures**

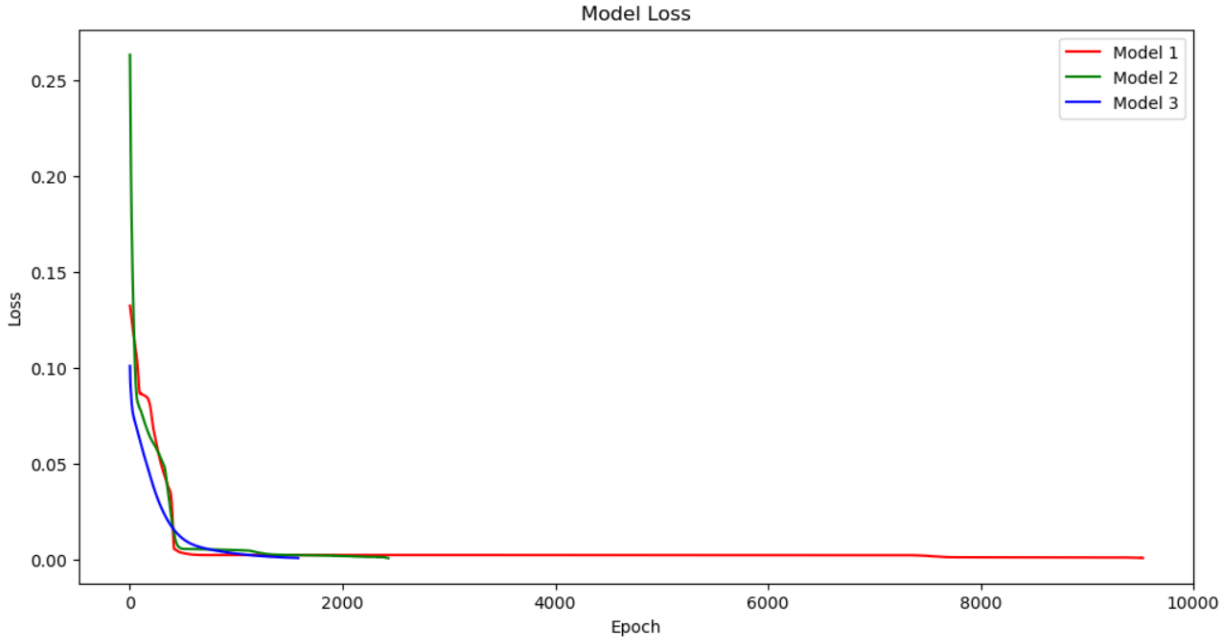
  Three different neural network architectures were implemented:

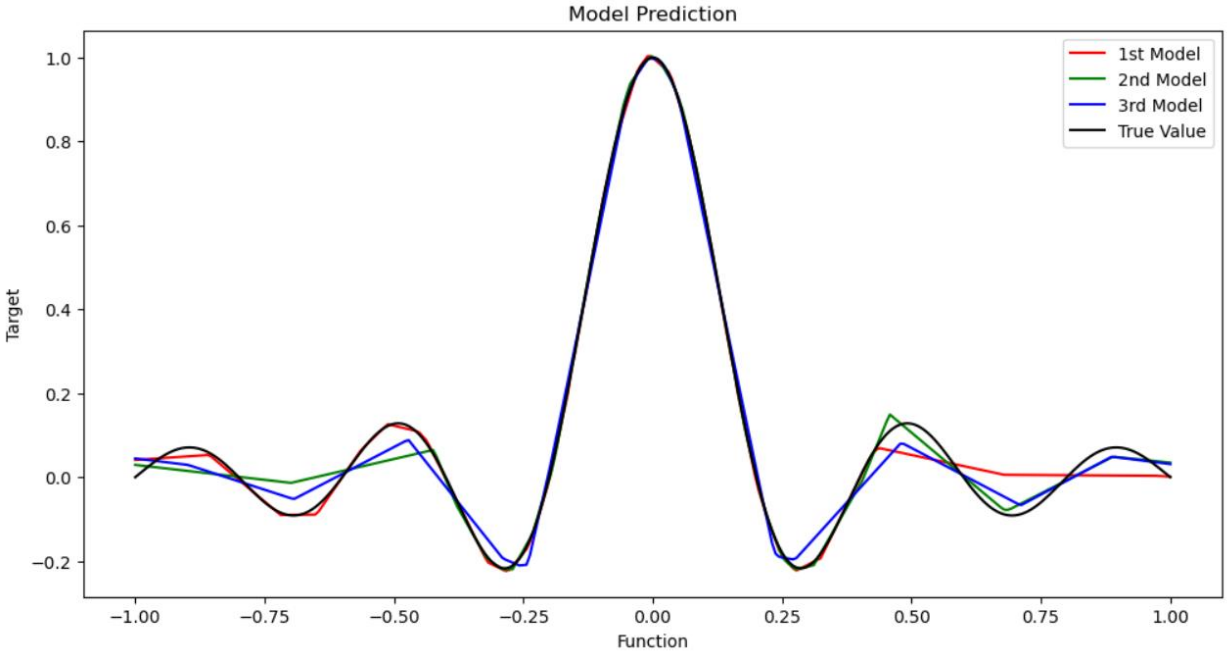
  Model 0 (CNNModel_Zero): A deep network with 7 hidden layers, each using Leaky ReLU activation.

  Model 1 (CNNModel_One): A moderately deep network with 4 hidden layers, also using Leaky ReLU activation.

  Model 2 (CNNModel_Two): A simpler network with a single hidden layer containing 190 neurons.

Function 1 Loss and Prediction for all the models:





Comments:

Loss Analysis for Function 1:

Model 1: Exhibited a steady decrease in loss, indicating effective learning.

Model 2: Also showed a decreasing trend, but with slightly higher loss values compared to Model 0.

Model 3: Demonstrated the fastest convergence, achieving low loss values quickly.

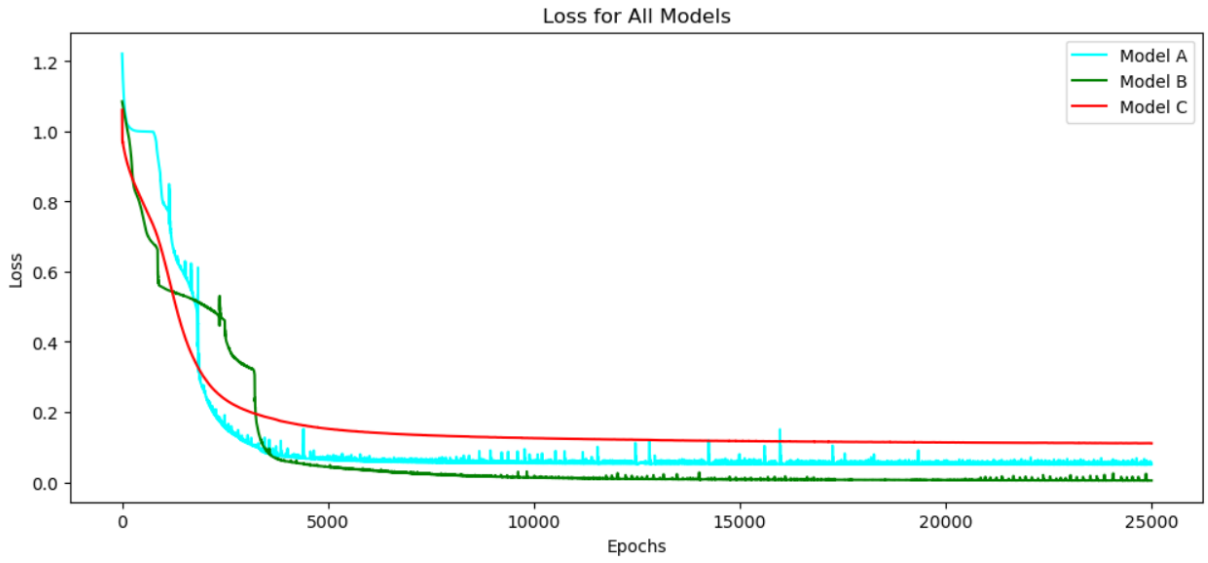
Prediction Analysis for Function 1:

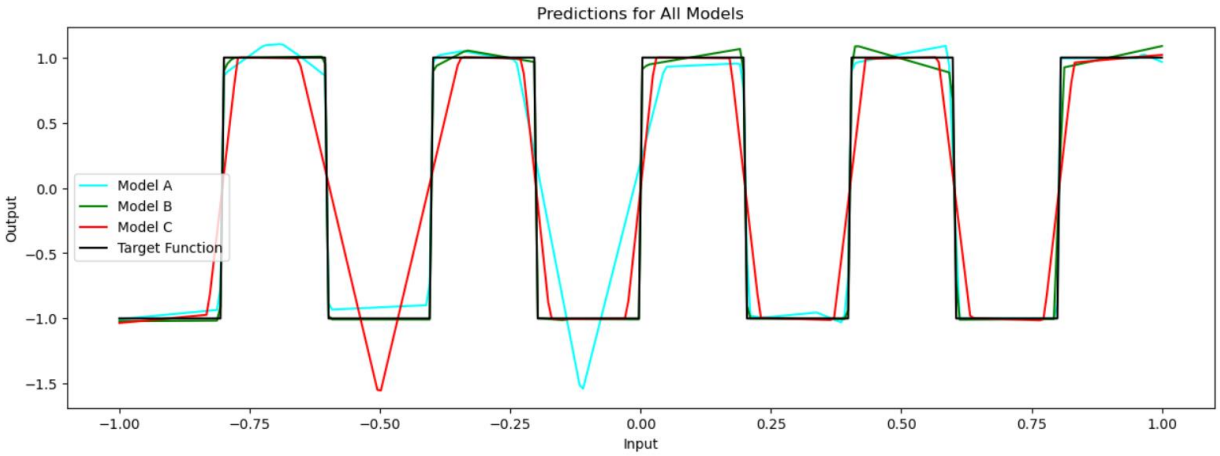
Model 1: Closely follows the true function, indicating good approximation.

Model 2: Also performs well but with slight deviations from the true function.

Model 3: While it captures the general trend, it shows more significant deviations from the true function compared to the other models.

Function 2 Loss and Prediction for all the models:





Comments:

Loss Analysis for Function 2:

Model A: Demonstrated a consistent decrease in loss, indicating effective learning.

Model B: Showed a similar trend but with slightly higher loss values.

Model C: Achieved the lowest loss values quickly, indicating efficient learning.

Prediction Analysis for Function 2:

Model A: Accurately captures the sign changes of the true function.

Model B: Also performs well, closely following the true function.

Model C: While it captures the general trend, it shows some deviations, particularly in the regions where the function changes sign.

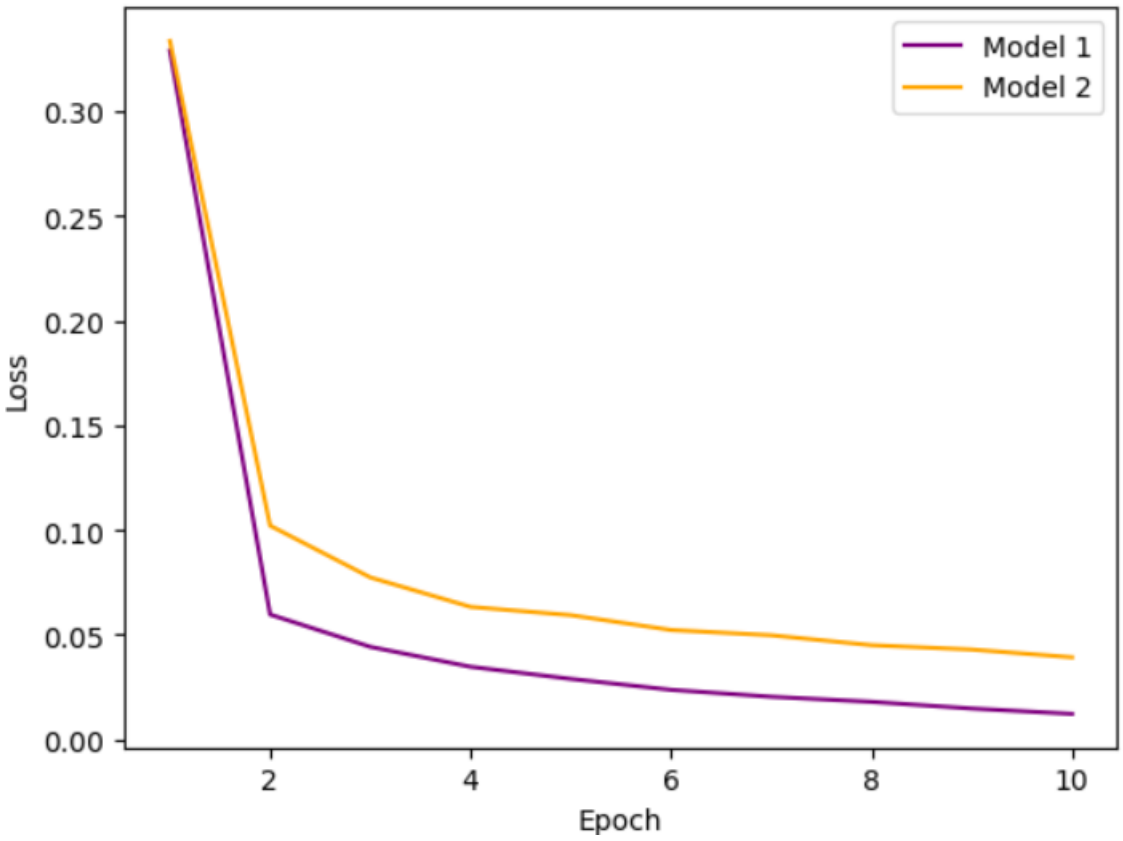
Train on actual task

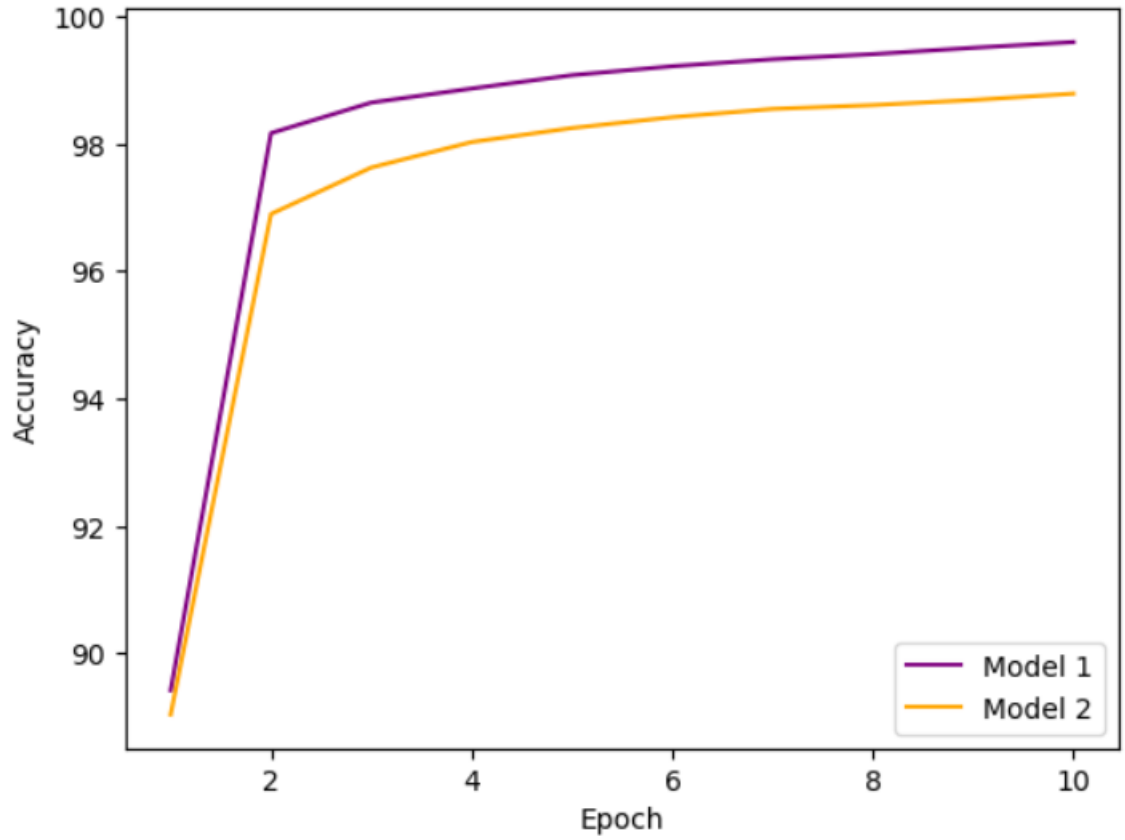
Settings

- **Task**: is to compare the performance of two different Convolutional Neural Network (CNN) architectures on the MNIST dataset.
- **Architecture**:
    - **SimpleCNN:** A basic CNN architecture with:

        - Two convolutional layers followed by ReLU activation and max pooling.
        - Two fully connected layers leading to the output layer.
    - **AdvancedCNN:** A more complex CNN architecture with:
        - Three convolutional layers, each followed by ReLU activation.
        - Dropout layers to prevent overfitting.
        - Two fully connected layers leading to the output layer.

MNIST Loss and Prediction for all the models:





Comments:

Loss Analysis for MNIST:

Model 1 (SimpleCNN): The loss decreases consistently, indicating that the model is learning.

Model 2 (AdvancedCNN): Also shows a decrease in loss, but at a faster rate, indicating more effective learning.

Prediction Analysis for MNIST:

Model A: Accurately captures the sign changes of the true function.

Model B: Also performs well, closely following the true function.

Model C: While it captures the general trend, it shows some deviations, particularly in the regions where the function changes sign.

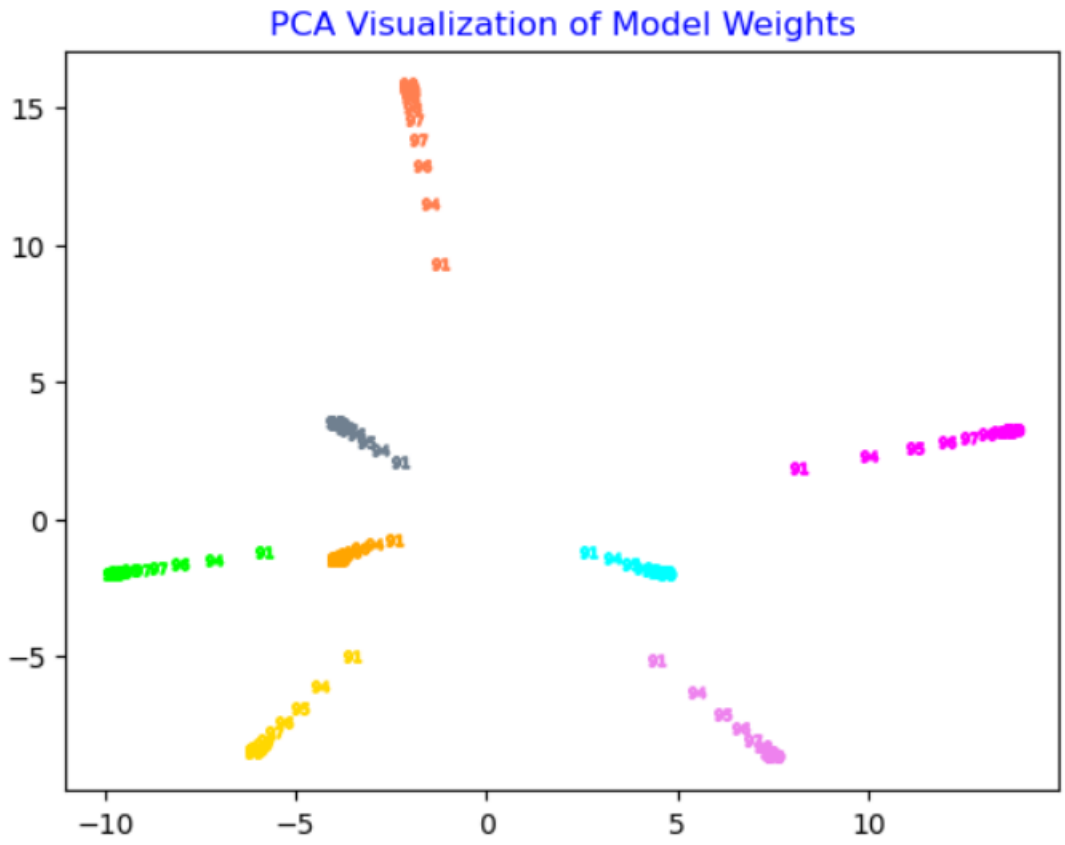
Optimization

Visualize the optimization process.

Settings

- **Architecture:** A feedforward neural network with three layers for MNIST classification and a separate regression model for function approximation.
- **Training Cycles**: The model is trained for 8 iterations, with parameters recorded for one specific layer and the entire model.
- **Optimizer**: Adam optimizer
- **Dimension Reduction Method**: Principal Component Analysis (PCA)

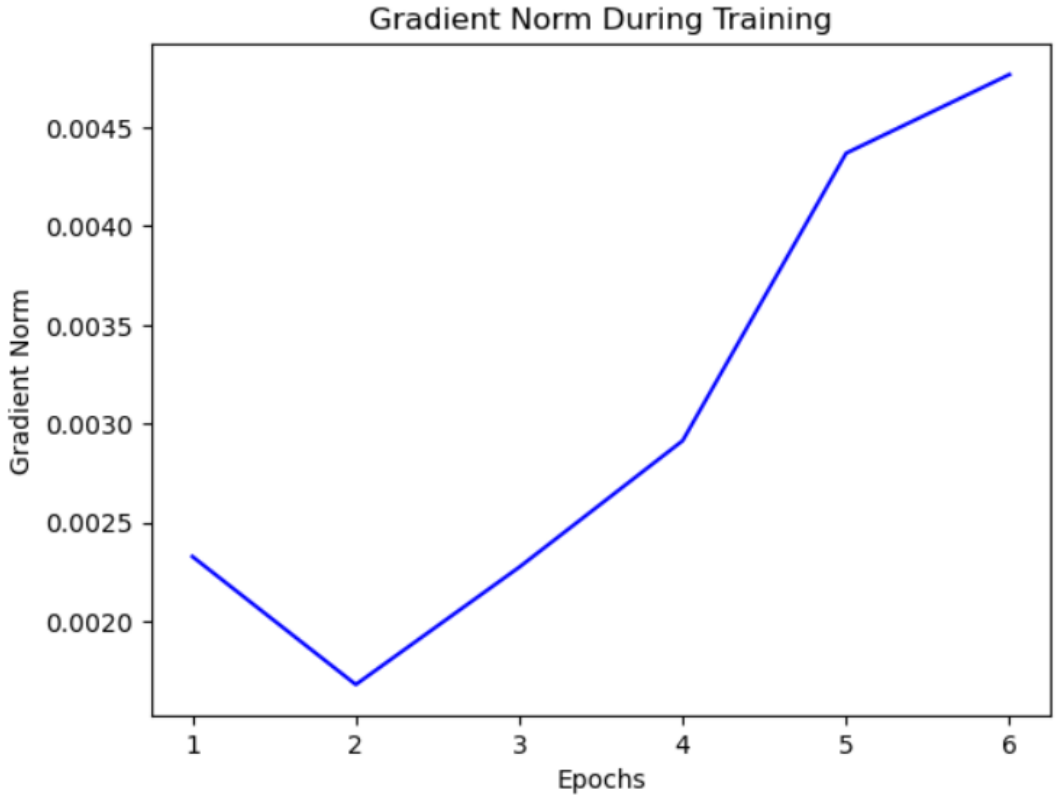
PCA Visualization of Model Weights:



Observe gradient norm during training.

The gradient norms are calculated during training to monitor the stability of the optimization process. The gradient norms are plotted against the training iterations to visualize their behavior.

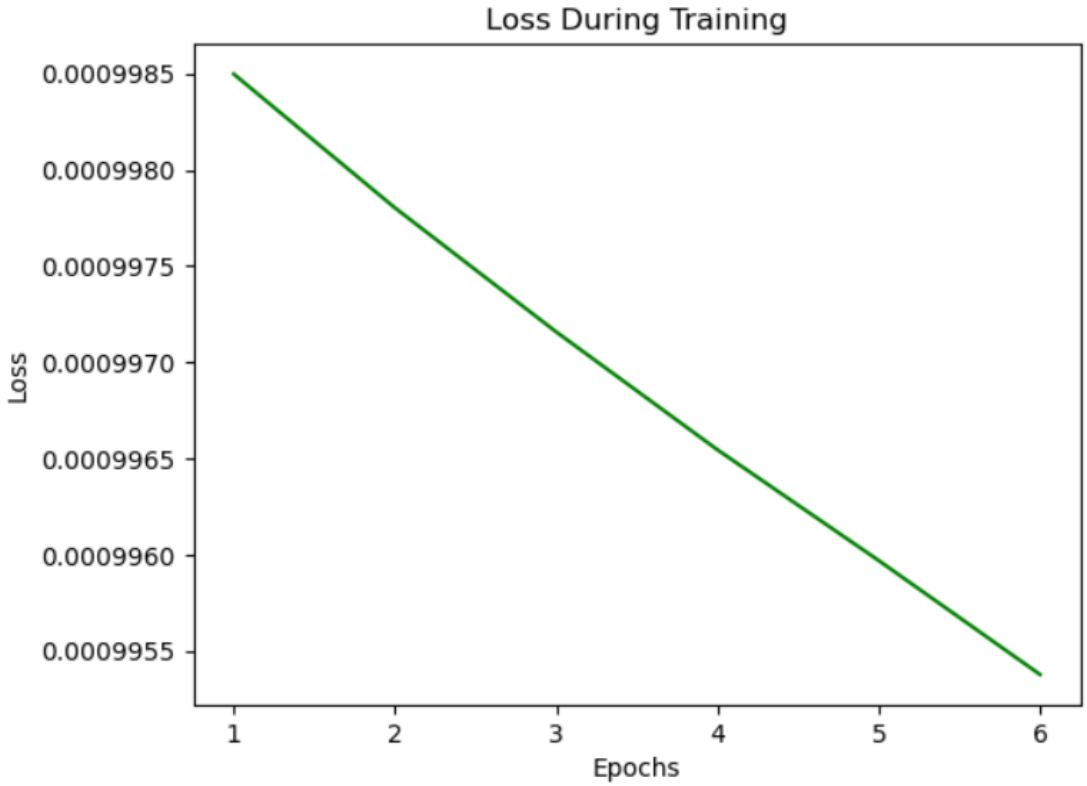
Gradient Norm During Training:



Comments:

The gradient norms fluctuate during training, indicating the model's learning dynamics. A decreasing trend in gradient norms generally corresponds to a decrease in loss.

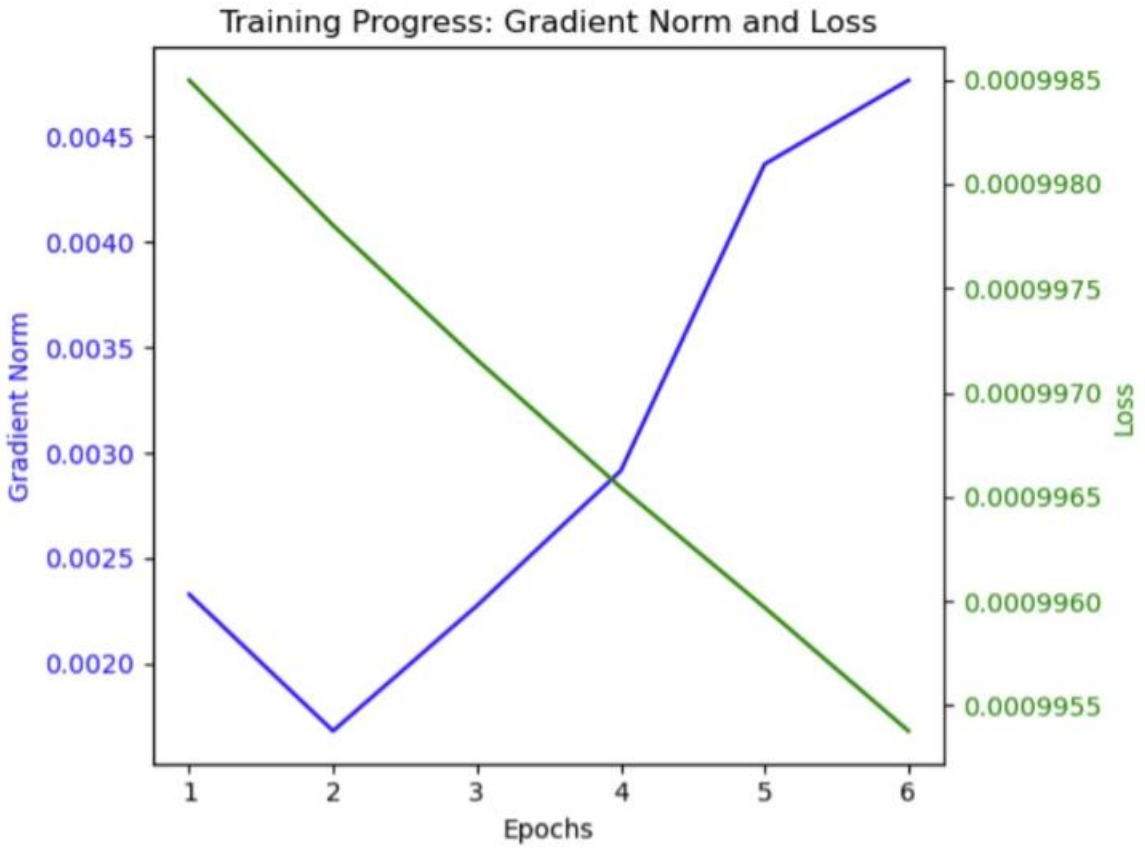
Loss During Training:



Comments:

The loss decreases steadily, indicating effective learning.

Gradient Norm and Loss:



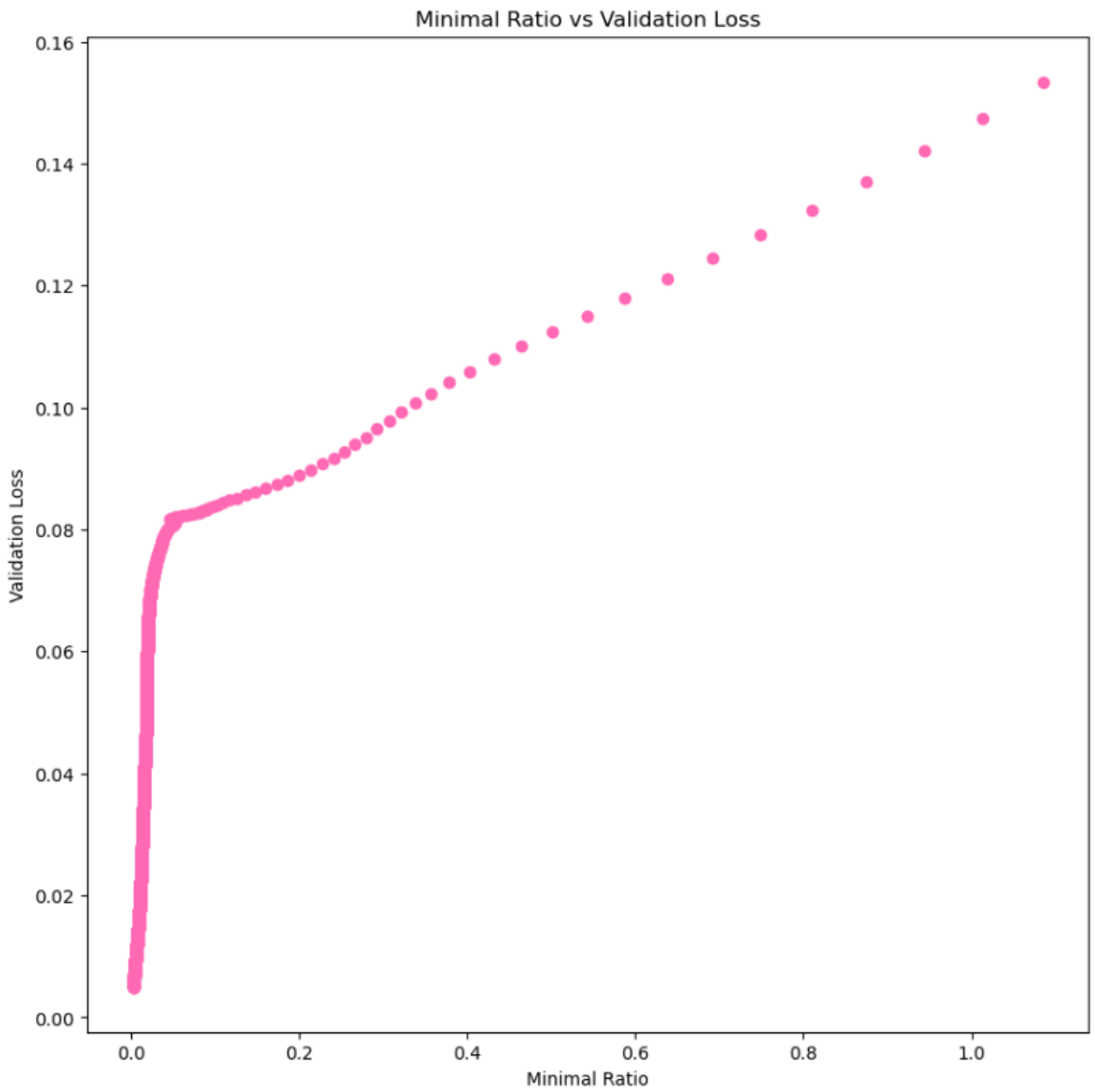
What happens when gradient is almost zero?

1. **Gradient Behavior:**
    - When the gradient is almost zero, it indicates that the model has reached a local minimum or is experiencing vanishing gradients. This can hinder further learning and optimization.
    - A zero gradient suggests that the model parameters are not being updated, which can lead to stagnation in training.

2. **Minimal Ratio Definition:**
    - The minimal ratio is defined as the ratio of the gradient norm to the loss. It provides insight into how effectively the model is learning relative to the magnitude of the gradients.

- The weights that yield a zero gradient are identified during the training process, and their corresponding minimal ratios are calculated.

Minimal Ratio vs Validation Loss:



Comments:

- The plot shows that as the minimal ratio decreases, the loss also tends to decrease, indicating that effective learning is occurring.

- A higher minimal ratio may suggest that the model is struggling to learn, potentially due to poor initialization or inappropriate learning rates.

Generalization

Can the network fit random labels?

Settings

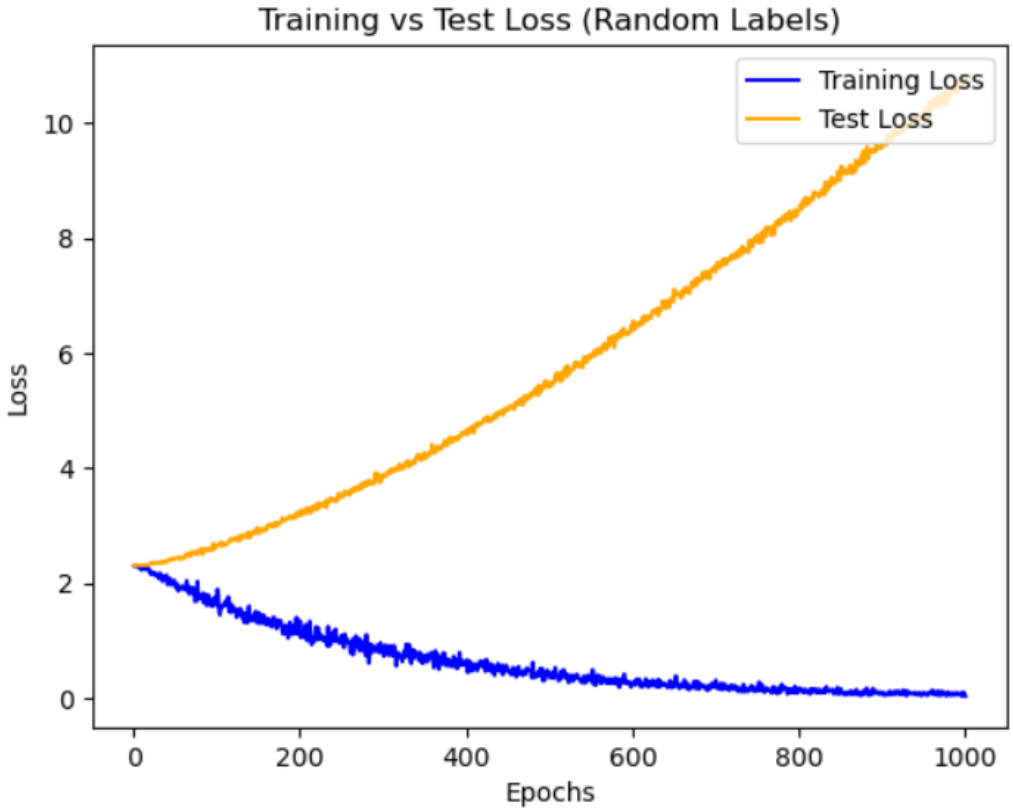
- **Task**: The task involves training a neural network on the MNIST dataset with random labels. The dataset includes 60,000 training and 10,000 test images of handwritten digits.
- **Learning Rate**: 0.0001
- **Optimizer**: Adam
- **Architecture**:

  The neural network consists of two fully connected layers:

  - Layer 1: 784 input units (flattened MNIST images) to 512 units, followed by a ReLU activation.
  - Layer 2: 512 units to 10 output classes (digits 0-9).

The experiment demonstrates that the neural network can fit random labels, as evidenced by the decreasing training loss over the epochs. However, this does not imply that the model is learning meaningful patterns; rather, it is simply minimizing the loss function based on the random labels provided.

Training and Testing Loss vs. Epochs



Comments: Training loss decreased, and testing loss increased over the epochs.

Number of parameters v.s. Generalization

Settings

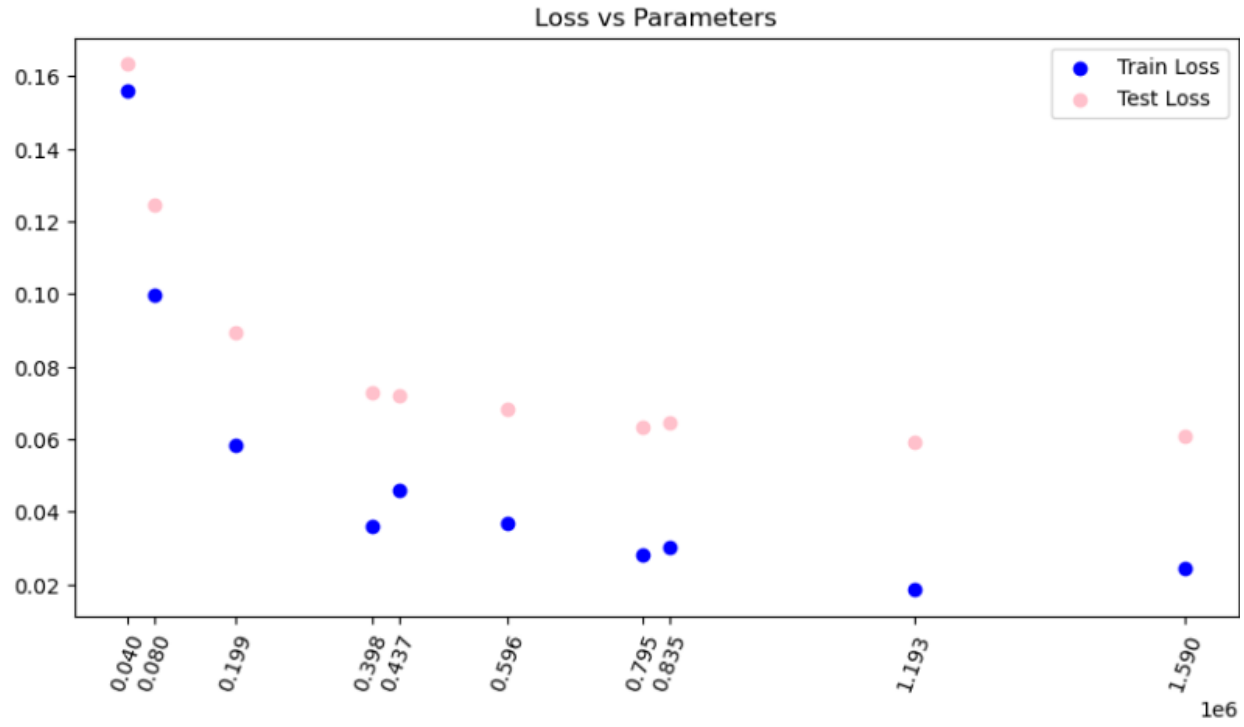
- **Task**: Classifying images of handwritten digits (0-9).
- **Training Data:** The training dataset consists of 60,000 images.
- **Testing Data:** The test dataset consists of 10,000 images.
- **Batch Size**: 600 for training and 100 for testing.
- **Optimizer**: Adam
- **Learning Rate:** 0.001

Model Architectures

Ten different feedforward neural network models were defined, each with varying numbers of neurons in the hidden layer:

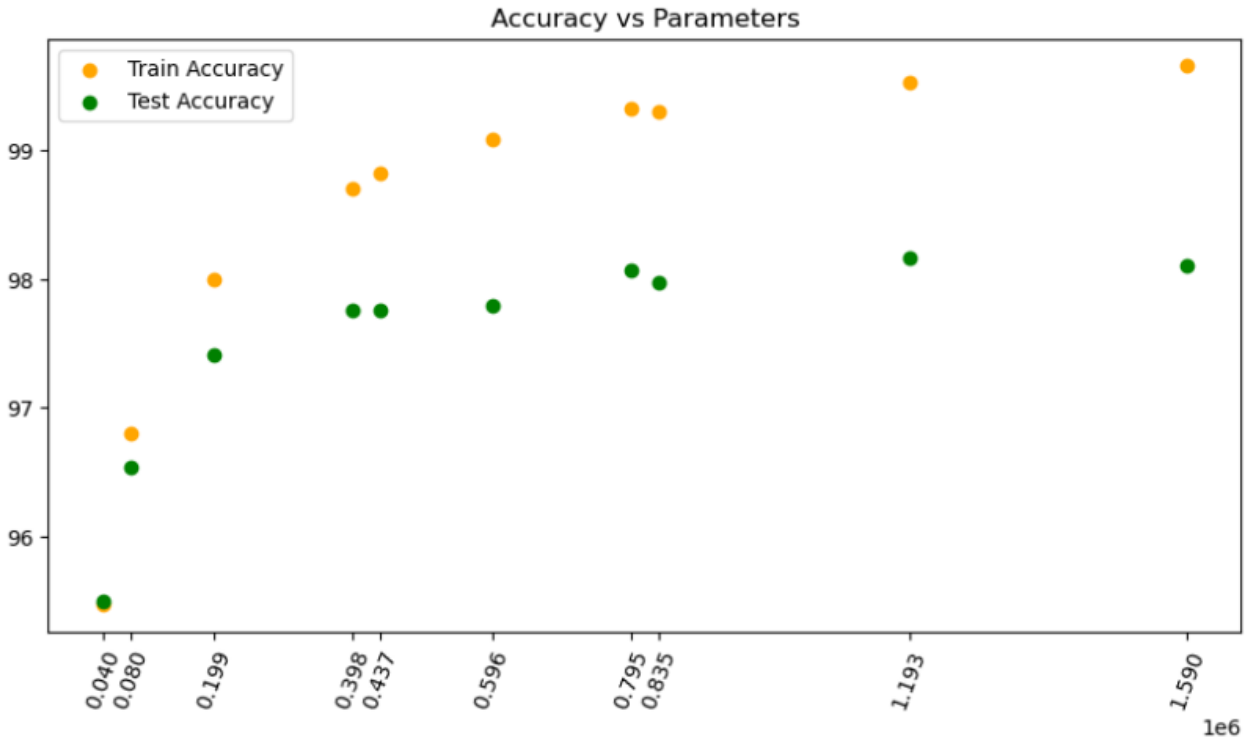
- **Model A**: 50 neurons
- **Model B**: 100 neurons
- **Model C**: 250 neurons
- **Model D**: 500 neurons
- **Model E**: 550 neurons
- **Model F**: 750 neurons
- **Model G**: 1000 neurons
- **Model H**: 1050 neurons
- **Model I**: 1500 neurons
- **Model J**: 2000 neurons

Loss vs Parameters



Comment: The number of parameters increases; the training loss tends to decrease. But the testing loss does not consistently decrease when parameters increase.

Accuracy vs Parameters



Comment: Training accuracy tends to improve with more parameters, reflecting the model's ability to fit the training data. The relationship between testing accuracy and the number of parameters is more intricate. Some models with a moderate number of parameters achieve strong accuracy, while others with a high number of parameters may underperform.

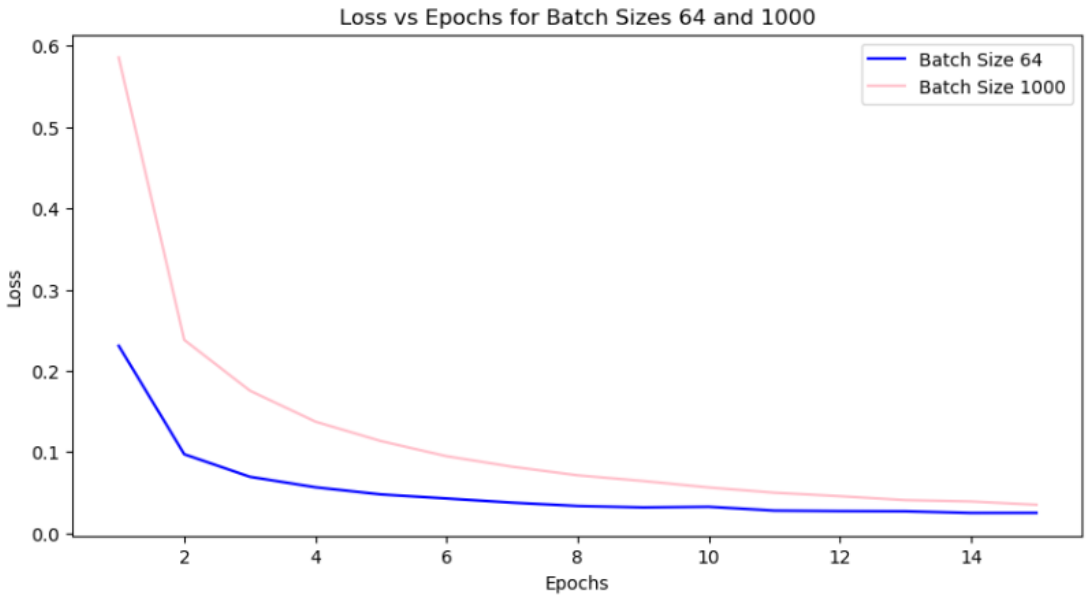
Flatness v.s. Generalization

Part1

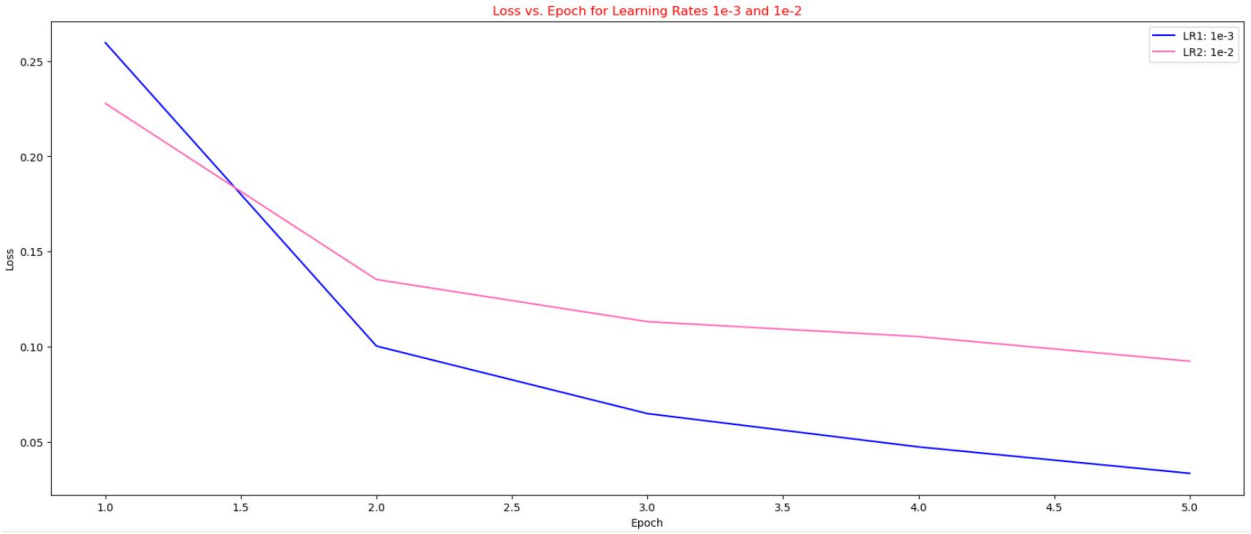
Settings

- **Task**: The task is to train two neural network models with different batch sizes on the MNIST dataset and analyze their performance.
- **Training Data:** The training dataset consists of 60,000 images.
- **Testing Data:** The test dataset consists of 10,000 images.
- **Batch Size**: Two models are trained with different batch sizes: 64 and 1000.
- **Optimizer**: Adam
- **Learning Rate:** 0.0015
- **Model Training:**
  Two models were trained with different batch sizes:
  Model with Batch Size 64:
  Training Loss: Recorded over 15 epochs.
  Training Accuracy: Calculated at the end of each epoch.
  Model with Batch Size 1000:
  Training Loss: Recorded over 15 epochs.
  Training Accuracy: Calculated at the end of each epoch.

Loss vs Epochs for different batches



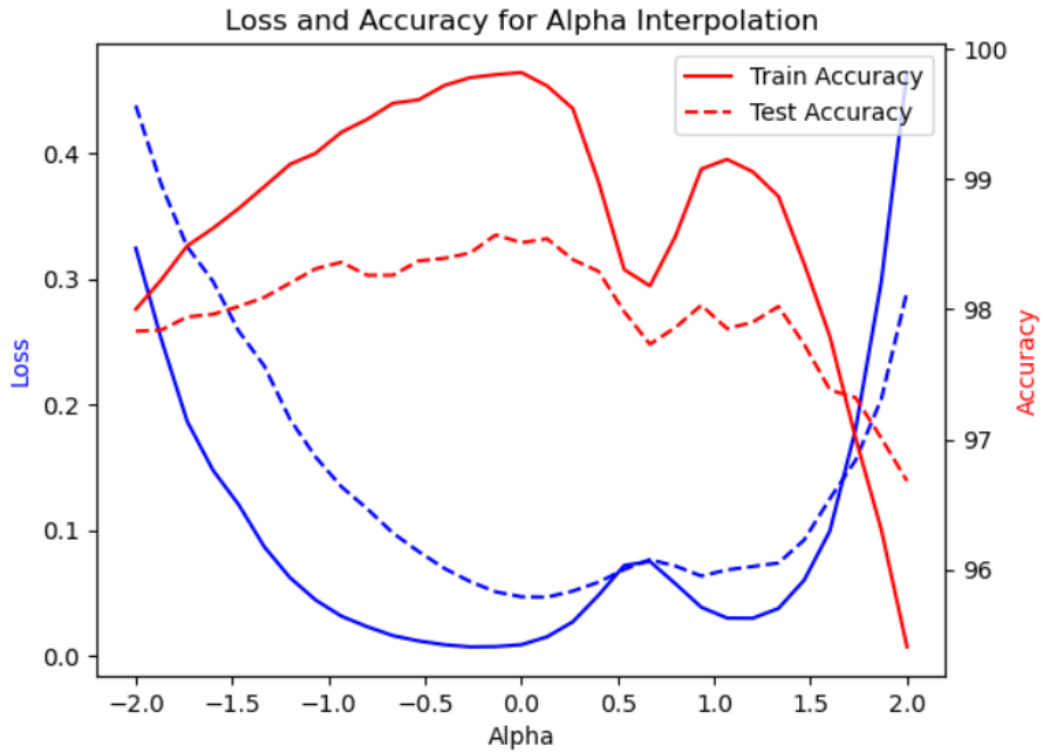
Loss vs Epoch for Learning Rates 1e-3 and 1e-2



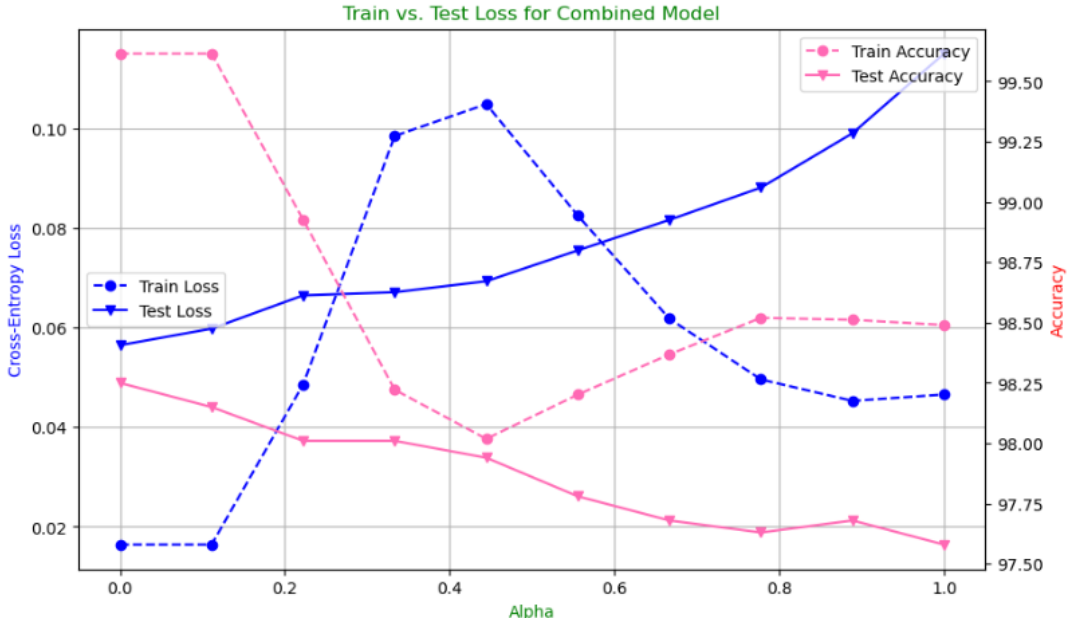
Comments:

- **Batch Size 64:** The training loss decreases steadily, indicating effective learning.
- **Batch Size 1000:** The training loss also decreases but at a slower rate, suggesting that larger batch sizes may lead to less frequent updates to the model parameters.

Loss and Accuracy for Alpha Interpolation



Train vs. Test Loss and Accuracy for Combined Model



Comments:

- **Train Loss:** The training loss shows a varying trend depending on the interpolation parameter α.
- **Test Loss:** The test loss also varies, indicating how the model's performance changes with different parameter combinations.
- **Train Accuracy:** The training accuracy fluctuates with αα, reflecting the model's ability to generalize based on the interpolated parameters.
- **Test Accuracy:** The test accuracy follows a similar trend, indicating the model's performance on unseen data.