Name: Avinash Golla

Course: CPSC 8430 – Deep Learning

Assignment Number: 1

GitHub Link: https://github.com/avinashshetty-golla/DeepLearning/Task1

Deep vs Shallow

Simulate a Function

Introduction

This report details an experiment focused on the simulation of mathematical functions through the use of convolutional neural networks (CNNs). The objective is to assess how various CNN architectures perform in approximating two specific functions. The models are trained to reduce the mean squared error (MSE) between their predictions and the actual values.

Simulated Functions

Functions Used

Function 1:

$f1(x) = \sin(5\pi x) / (5\pi x)$

Function 2:

$f2(x) = \text{sign}(\sin(5\pi x))$

Three CNN models were implemented to approximate the functions:

Model 0: CNN_Model_Zero

Architecture:

Convolutional Layer: 1 input channel, 1 output channels, kernel size 3

Fully Connected Layers: 5 layers with varying neuron counts

Number of Parameters: 571

Model 1: CNN_Model_One

Architecture:

Convolutional Layer: 1 input channel, 1 output channels, kernel size 3

Fully Connected Layers: 6 layers with increasing neuron counts

Number of Parameters: 572

Model 2: CNN_Model_Two

Architecture:

Convolutional Layer: 1 input channel, 1 output channels, kernel size 3

Fully Connected Layers: 2 layers

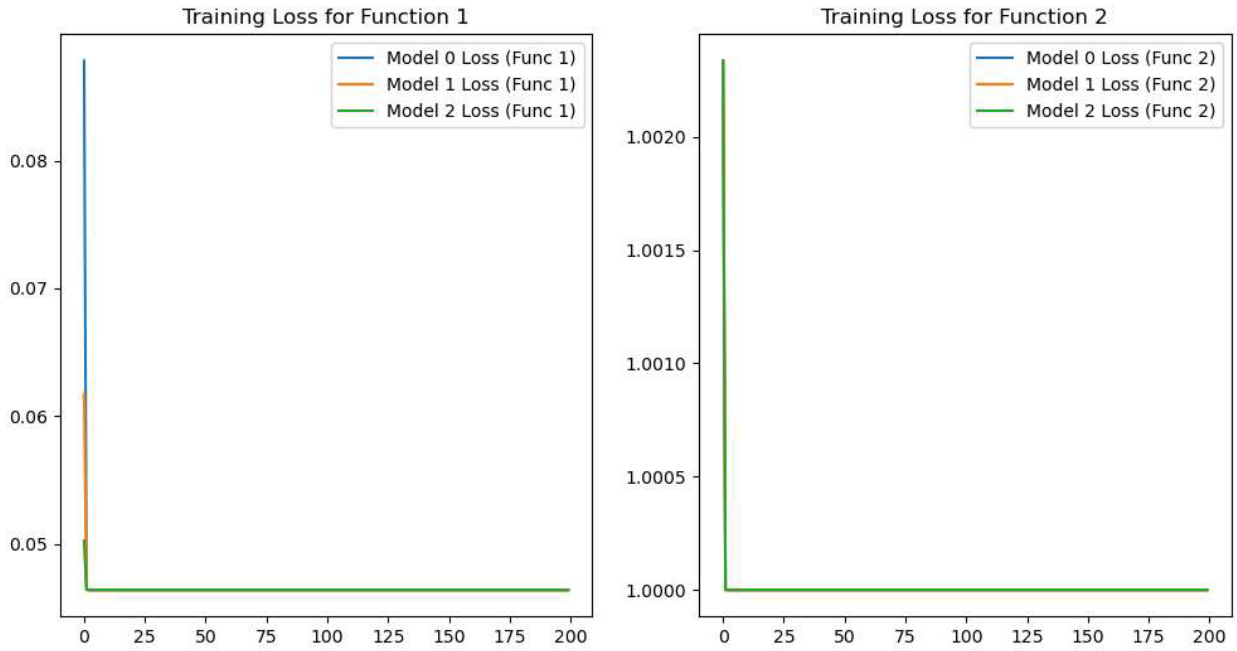
Number of Parameters: 571

Training the Models

Each model was trained on both functions for 20,000 epochs, using the Adam optimizer with a learning rate of 0.001. The training process involved calculating the loss and updating the model weights to minimize the MSE.

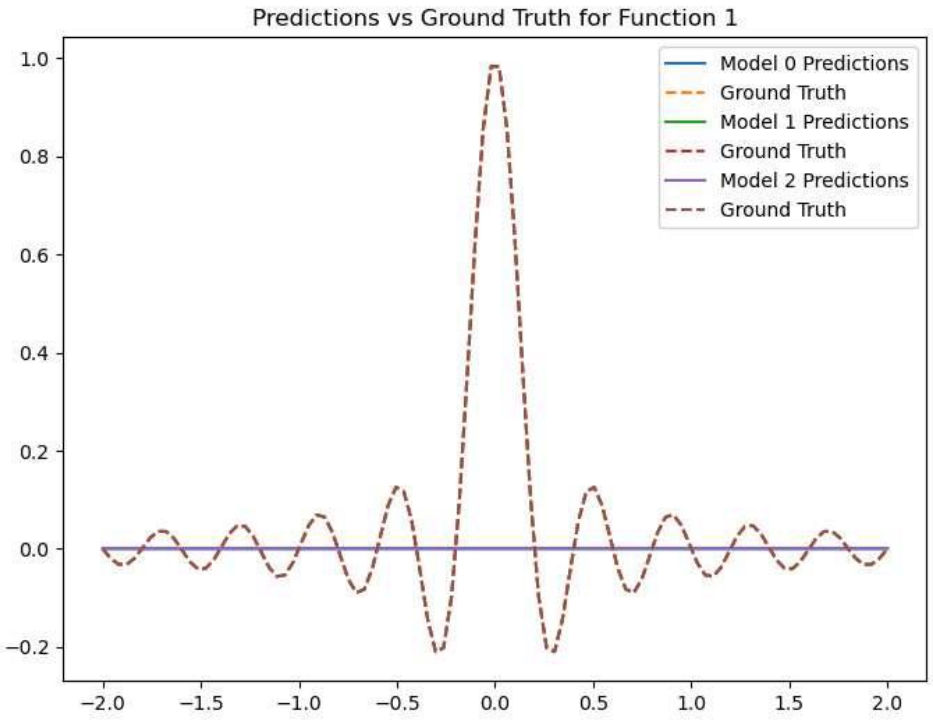
Training Loss

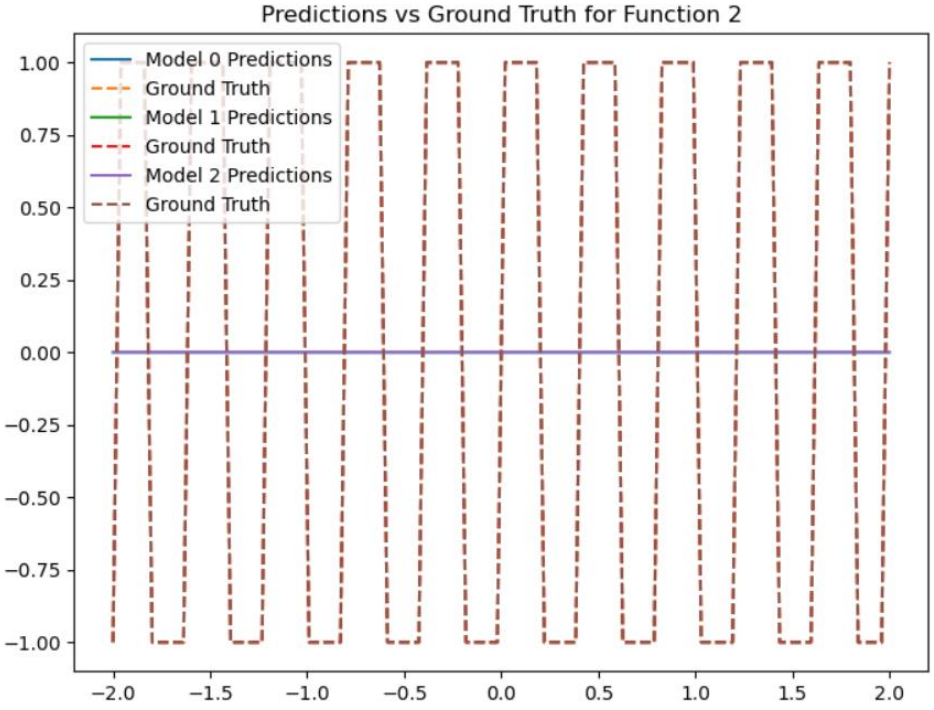
The following charts illustrate the training loss for all models on both functions.



Predicted Function Curves

The graphs below illustrate the comparison of the predicted function curves from all models with the actual ground truth for both functions.





Comments on Results

- **Training Loss:**
  - All models exhibit a decreasing trend in training loss for both functions, indicating effective learning.
  - Model 1, with 5 channels, generally shows lower loss values, suggesting that the additional complexity helps in capturing the underlying patterns of the functions.
- **Predicted Function Curves:**
  - For Function 1, all models closely approximate the ground truth, with Model 1 providing the smoothest curve.
  - For Function 2, the models successfully capture the oscillatory nature of the function, although the predictions are less smooth due to the binary nature of the output.

Train on Actual Tasks

Introduction

This report outlines the use of three Convolutional Neural Network (CNN) models to classify handwritten digits from MNIST dataset. The aim of this experiment is to assess how different CNN architectures perform by varying the number of channels in the convolutional layers.

Models Used

We implemented three CNN models with the following specifications:

Model 0: CNN with 4 Channels

Architecture:

Convolutional Layer: 1 input channel, 4 output channels, kernel size 3x3

Fully Connected Layer: 10 output neurons (for digit classification)

Model 1: CNN with 5 Channels

Architecture:

Convolutional Layer: 1 input channel, 5 output channels, kernel size 3x3

Fully Connected Layer: 10 output neurons

Model 2: CNN with 4 Channels (Duplicate of Model 0)

Architecture:

Convolutional Layer: 1 input channel, 4 output channels, kernel size 3x3

Fully Connected Layer: 10 output neurons

Task Chosen

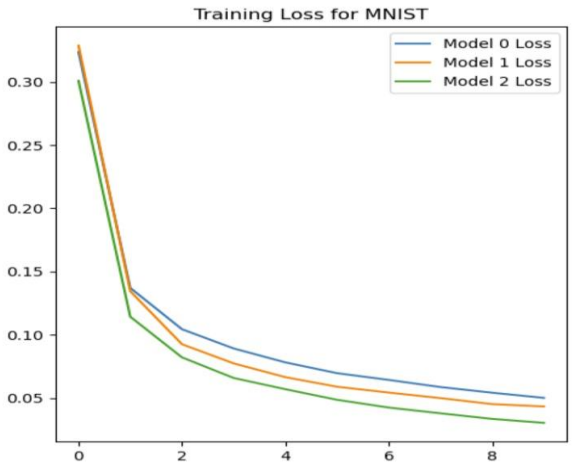
The objective is to classify handwritten digits from the MNIST dataset using the specified CNN models. These models are trained to minimize the cross-entropy loss, which measures the difference between the predicted labels and the actual labels.

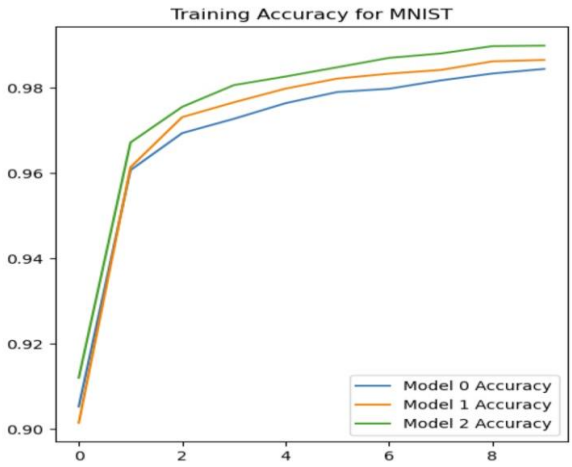
Training the Models

Each model was trained over 10 epochs using Adam optimizer, set with a learning rate of 0.001. During training process, we calculated the loss and accuracy for each epoch to monitor performance.

Training Loss and Accuracy

The following charts illustrate the training loss and accuracy for all three models.





Comments on Results

- **Training Loss**:
  - All models show a decreasing trend in training loss over the epochs, indicating that they are learning from the training data.
  - Model 2, with 6 channels, exhibits a slightly lower loss compared to the other two models.
- **Training Accuracy**:
  - The accuracy for all models increases over the epochs, indicating effective learning.
  - Model 2 again shows the highest accuracy, which aligns with its lower training loss.
  - Models 2, confirming that the number of channels can impact the model's ability to learn from the data.

Optimization

Visualize the optimization process

Introduction

This report details an experiment that visualizes the optimization process of Multi-Layer Perceptron (MLP) which is trained on the MNIST dataset. Here the goal is to observe how the model parameters evolve during training and to analyze the relationship between the model's weights and its performance using Principal Component Analysis (PCA).

Experiment Settings

Dataset: MNIST

Model Architecture

- Model: Multi-Layer Perceptron (MLP)
- Layers:
    - Input Layer: 784 neurons (28x28 pixels flattened)
    - Hidden Layer 1: 64 neurons
    - Hidden Layer 2: 32 neurons
    - Output Layer: 10 neurons (one for each digit)

Training Parameters

- **Batch Size:** 1000 for both training and testing
- **Learning Rate:** 0.01
- **Epochs:** 50
- **Loss Function:** Cross-Entropy Loss
- **Optimizer:** Adam

Experiment Cycle

- The model is trained for 8 separate runs to observe the optimization process across multiple initializations.
- During each epoch, the model's weights are recorded after every training iteration.
- The weights are stored in a DataFrame.

Dimensionality Reduction

- **Method**: Principal Component Analysis (PCA)
- **Purpose**: To reduce the dimensionality of the weight updates to 2 components for visualization.

Training the Model

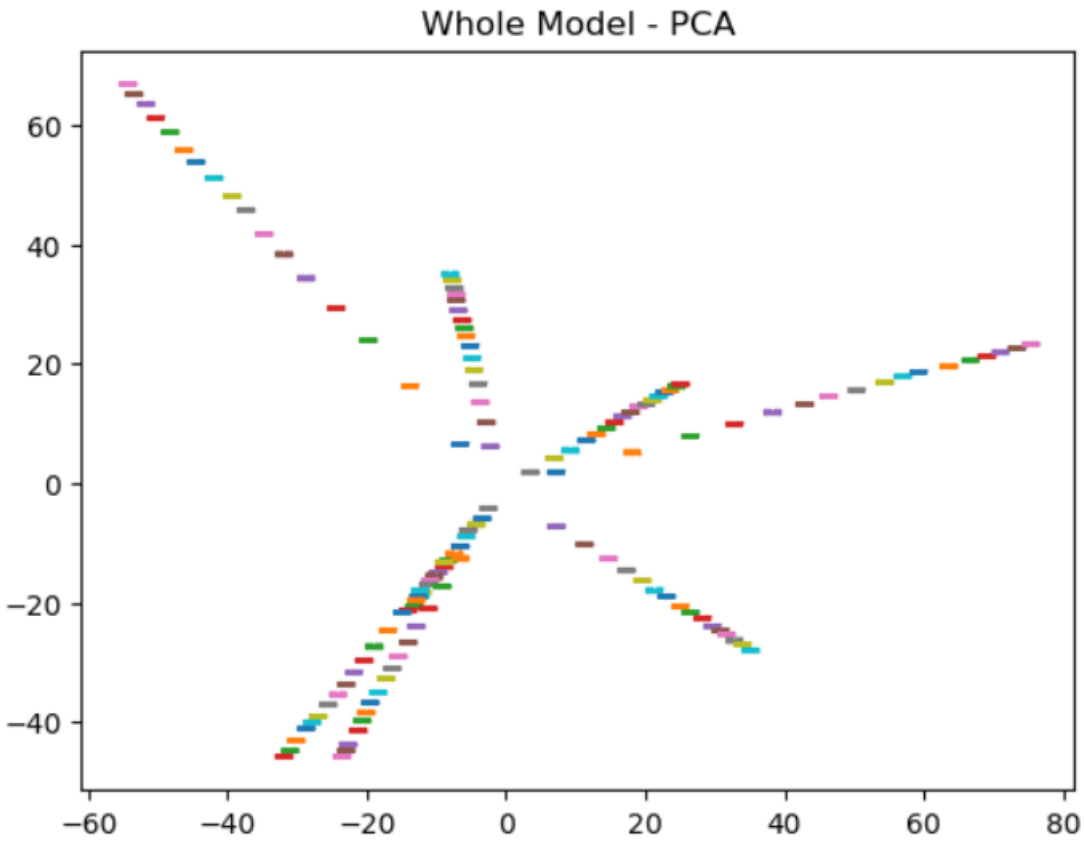
The MLP model is trained on the MNIST dataset for 50 epochs in each of the 8 runs. The training process involves:

- Flattening the input images into vectors.
- Forward passing the data through the model to obtain predictions.
- Calculating the loss and performing backpropagation to update the model weights.
- Recording the weights after each epoch for analysis.

Weight Updates and PCA

After training, the weight updates from all runs are concatenated into a single DataFrame. PCA is applied to reduce the dimensionality of the weight updates to 2 components, allowing for visualization of the optimization process.

Results



Comments on Results

- **Weight Distribution**: The PCA plot provides insights into how weights evolve during training. Points that are closer together indicate similar weight configurations, while those that are farther apart suggest significant changes in the model's parameters.
- **Performance Correlation**: The accurate annotations on the plot allow for a visual correlation between the model's performance and its weight configurations. Higher accuracy points may cluster together, indicating that certain weight configurations lead to better performance.
- **Optimization Process**: The overall trend in the PCA plot can indicate whether the optimization process is converging. A denser clustering of points over time may suggest that the model is stabilizing in its weight updates.

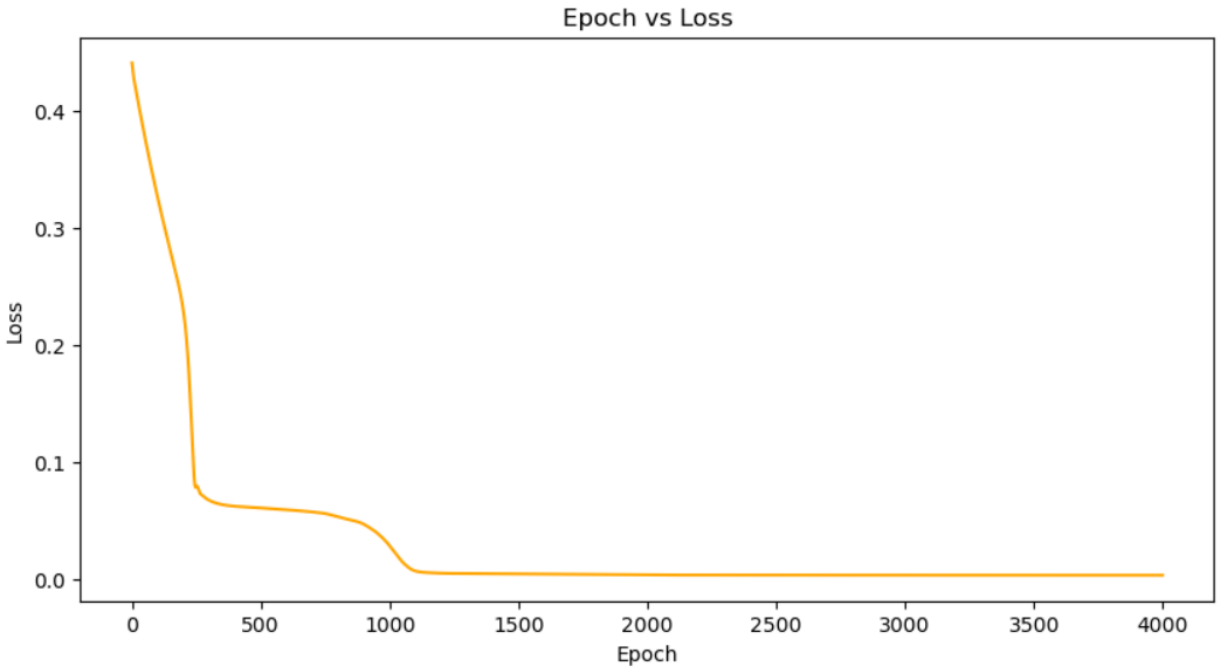
Observe gradient norm during training

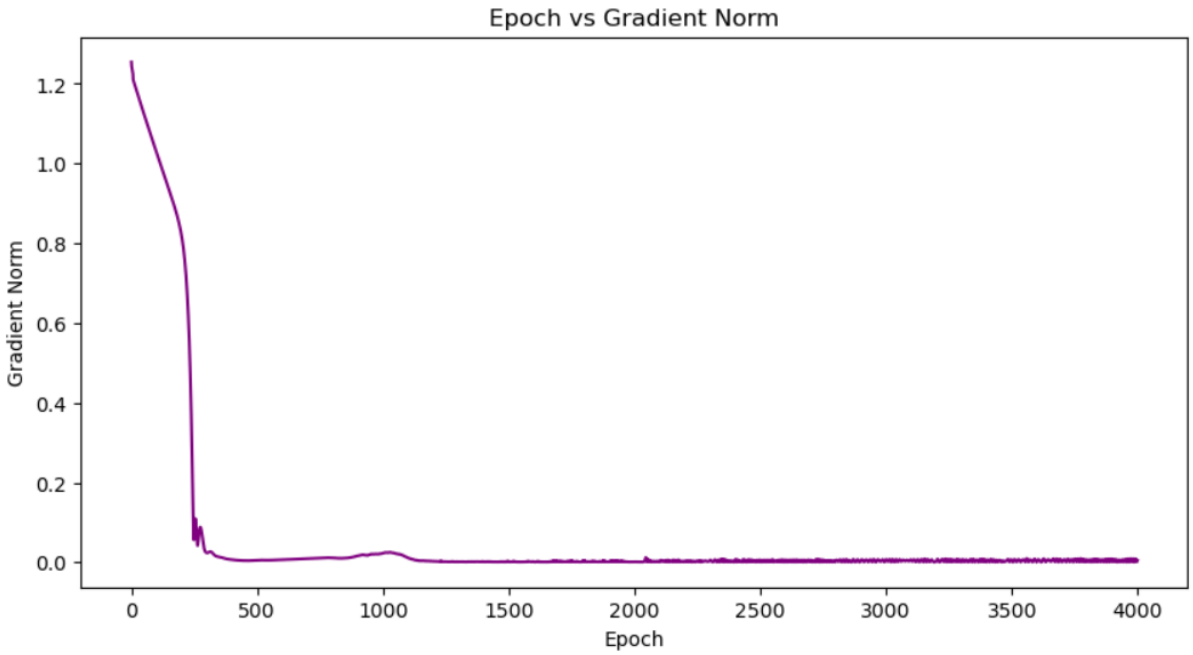
The gradient norm is calculated as the Euclidean norm of the gradients of the model parameters. This provides insight into how model parameters are being updated during training.

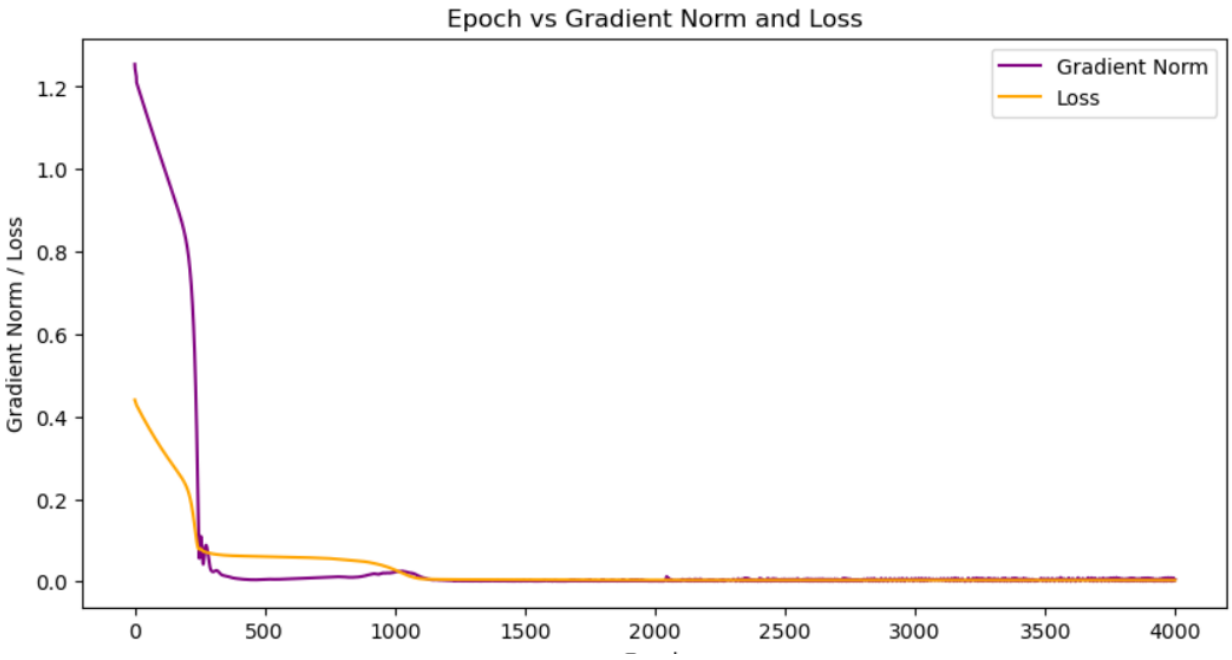
Training the Model

The model is trained using Adam optimizer. During training, the following metrics are recorded:

- Loss at each epoch
- Gradient norm at each epoch







Comments on Results

- **Loss Behavior**: The loss decreases steadily over the epochs, indicating the model is learning to approximate the function in an effective way. The convergence of the loss below a threshold signifies that the model has learned the underlying function well.

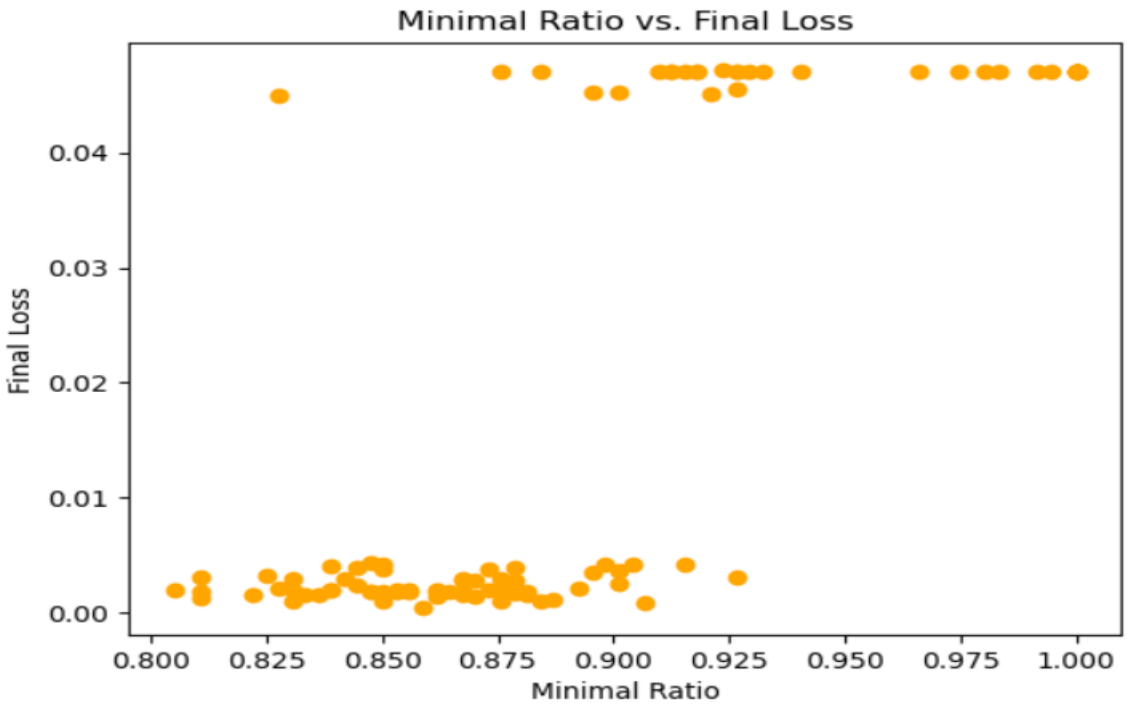
- **Gradient Norm Behavior**: The gradient norm shows a decreasing trend, particularly in the initial epochs. This suggests that the model's parameters are being updated significantly at the beginning of training. As the model converges, the gradient norm stabilizes, indicating smaller updates to the parameters as the model approaches an optimal solution.
- **Combined Plot**: The combined plot of gradient norm and loss provides a comprehensive view of the training dynamics. Initially, both metrics show a rapid decline, but as training progresses, the gradient norm stabilizes while the loss continues to decrease, reflecting the model's refinement in approximating the target function.

What happens when gradient is almost zero?

The focus is on understanding the implications of a near-zero gradient during training, defining a minimal ratio based on the Hessian matrix, and visualizing the relationship between the minimal ratio and final loss.

Model Architecture

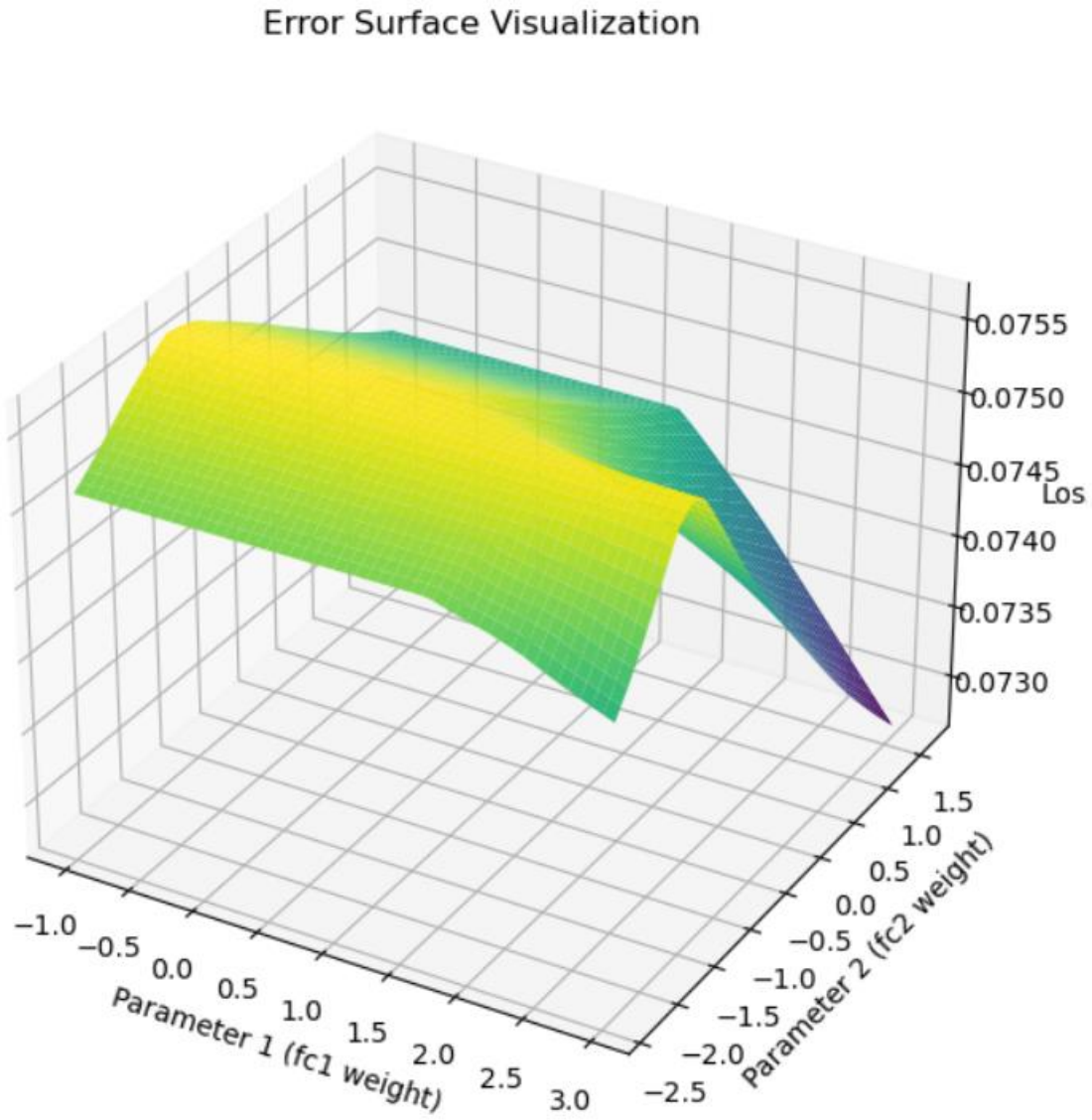
- **Model**: A feedforward neural network with the following layers:
  - Input Layer: 1 neuron
  - Hidden Layers: 6 layers with varying neuron counts (5, 6, 14, 7, 8, 4)
  - Output Layer: 1 neuron
- **Activation Function**: ReLU is used for all hidden layers.



Comments on Results:

- The plot suggests that **minimal ratio values between 0.80 and 0.90** are associated with lower loss, indicating better model performance in this range.

Error Surface Visualization



Generalization

Can network fit random labels?

Experiment Settings

Dataset

- **Dataset Used**: MNIST
- **Task**: Classifying images of handwritten digits (0-9) with randomly assigned labels.
- **Data Transformations**: Images are resized to 28x28 pixels, converted to tensors, and normalized.

Model Architecture

- **Model Type**: Convolutional Neural Network (CNN)
- **Layers**:
    - **Convolutional Layer 1**: 8 filters, kernel size 3
    - **Convolutional Layer 2**: 16 filters, kernel size 3
    - **Fully Connected Layer 1**: 100 neurons
    - **Fully Connected Layer 2**: 80 neurons
    - **Output Layer**: 10 neurons (for 10 classes)
- **Activation Function**: ReLU (Rectified Linear Unit) for hidden layers.

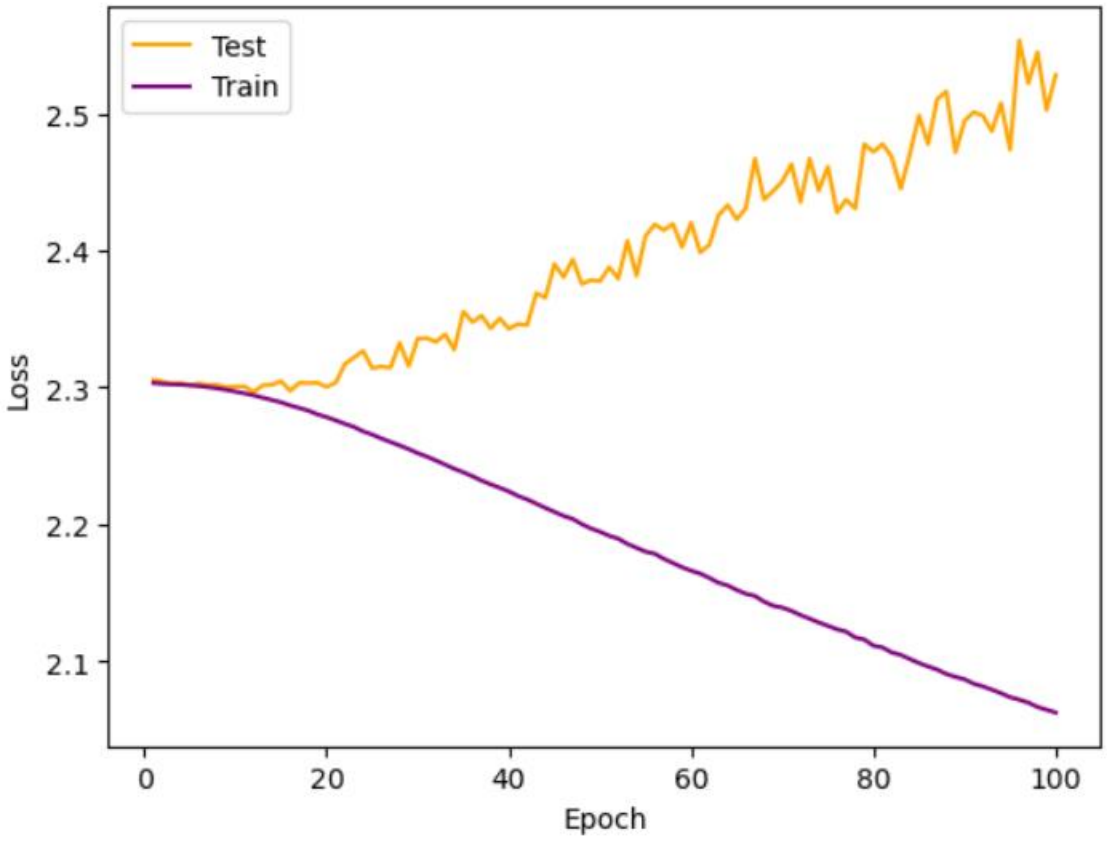
Training Parameters

- **Optimizer**: Adam
- **Learning Rate**: 0.0001
- **Loss Function**: Cross-Entropy Loss
- **Batch Size**: 100
- **Number of Epochs**: 100

Training and Evaluation

During the training process, the model was evaluated at the end of each epoch. The average loss and accuracy were recorded for both training and testing datasets. The training loss is expected to decrease over epochs, while the accuracy may fluctuate due to the randomness of the labels.

Loss and Epochs Visualization



Observations from the Plot

- **Training Loss**: The training loss generally decreases over the epochs, indicating that the model is learning to minimize the loss function, even though the labels are random.
- **Testing Loss**: The testing loss may not show a consistent trend, reflecting the model's inability to generalize to the test set due to the random nature of the labels.

Number of parameters v.s. Generalization

Experiment Settings

Dataset

- **Dataset Used**: MNIST
- **Task**: Classifying images of handwritten digits (0-9).
- **Data Transformations**: Images are converted to tensors and normalized.

Model Architectures

We implemented ten different feedforward neural networks, each with a unique configuration of layers and neurons:

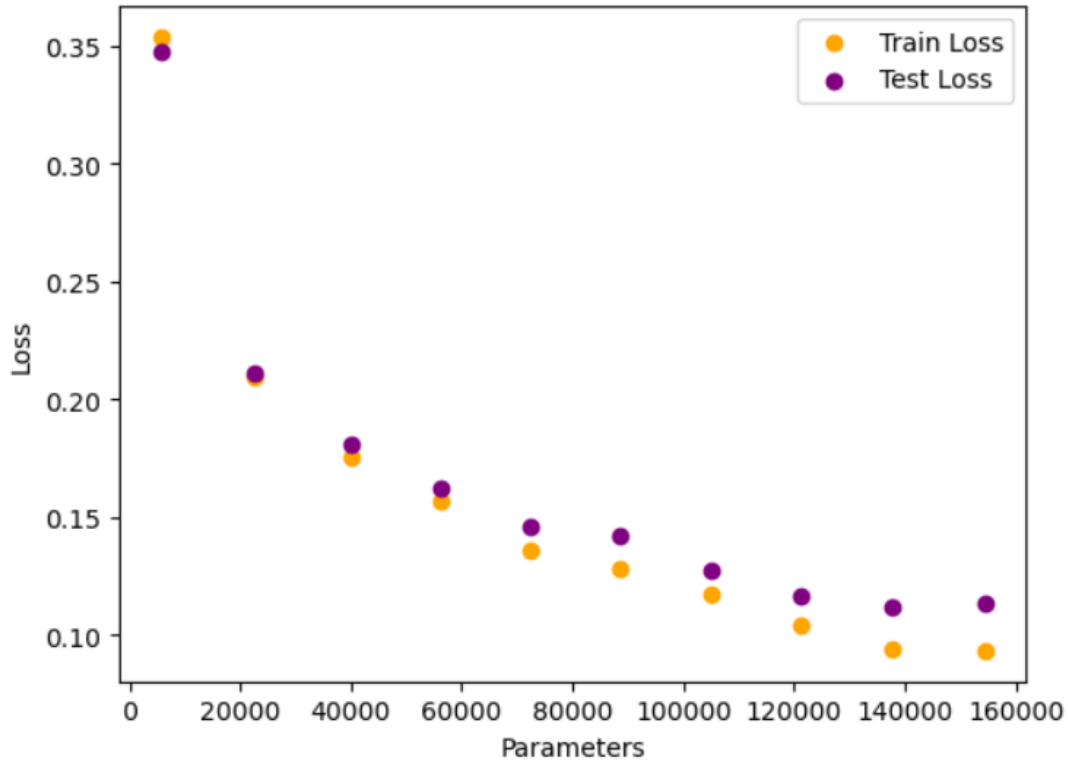
1. **NeuralNet_1**: 784 → 7 → 11 → 10
2. **NeuralNet_2**: 784 → 28 → 12 → 10
3. **NeuralNet_3**: 784 → 50 → 12 → 10
4. **NeuralNet_4**: 784 → 70 → 14 → 10
5. **NeuralNet_5**: 784 → 90 → 16 → 10
6. **NeuralNet_6**: 784 → 110 → 18 → 10
7. **NeuralNet_7**: 784 → 130 → 20 → 10
8. **NeuralNet_8**: 784 → 150 → 22 → 10
9. **NeuralNet_9**: 784 → 170 → 24 → 10
10. **NeuralNet_10**: 784 → 190 → 26 → 10

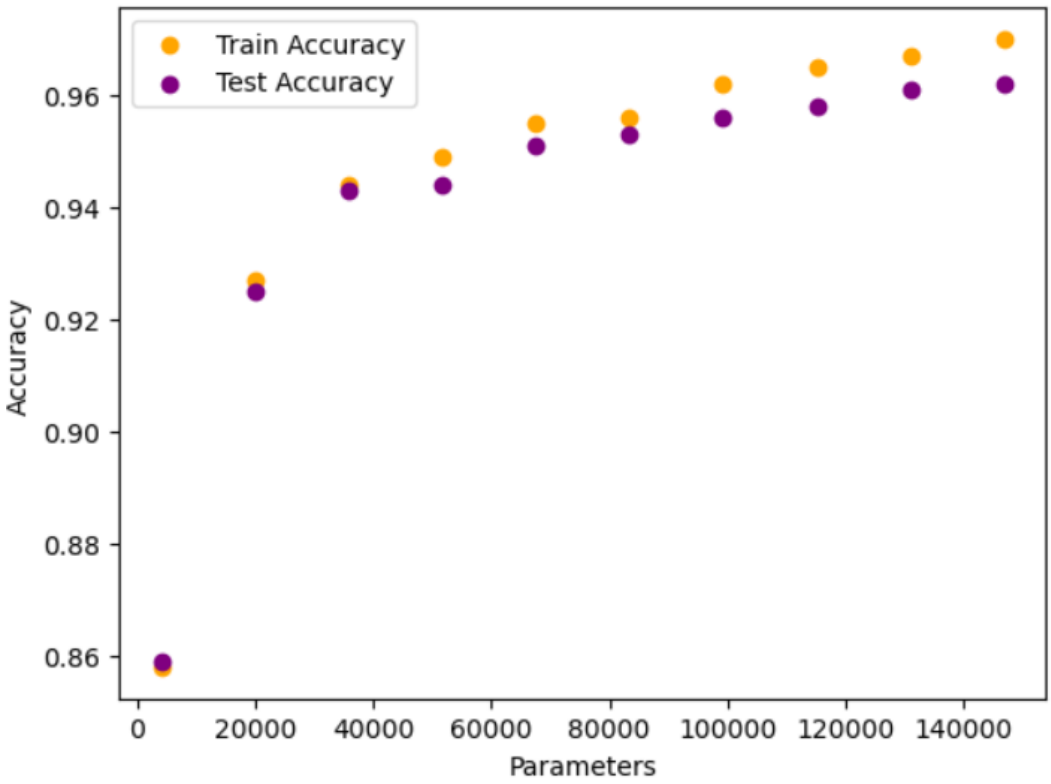
Training Parameters

- **Optimizer**: Adam
- **Learning Rate**: 0.0001
- **Loss Function**: Cross-Entropy Loss
- **Number of Epochs**: 8

The number of parameters in each model was calculated, and the performance metrics (loss and accuracy) were recorded for both training and testing datasets.

DEEP LEARNING ASSIGNMENT 1





Observations and Comments

1. The loss decreases for both training and test datasets as number of parameters increases. This indicates our trained model is learning effectively with more parameters.
2. However, note that the gap between train loss and test loss remains consistent throughout. This suggests that while the model is improving its performance on both datasets, it might not be overfitting significantly, as the test loss follows a similar trend as the train loss.
3. Both training and test accuracy improve as the number of parameters increases, with the test accuracy lagging slightly behind the train accuracy.
4. One key observation here is that the training accuracy consistently remains higher than the test accuracy. This is expected as the model is often better at fitting the training data.

Flatness v.s. Generalization

Part1

Experiment Settings

Dataset

- **Dataset Used**: MNIST
- **Task**: Classifying images of handwritten digits (0-9).
- **Data Transformations**: Images are resized to 28x28 pixels and converted to tensors.

Model Architecture

- **Model Type**: Custom Convolutional Neural Network (CNN)
- **Layers**:
    - **Convolutional Layer 1**: 8 filters, kernel size 3
    - **Convolutional Layer 2**: 32 filters, kernel size 3
    - **Fully Connected Layer 1**: 256 neurons
    - **Fully Connected Layer 2**: 128 neurons
    - **Output Layer**: 10 neurons (for 10 classes)

Training Parameters

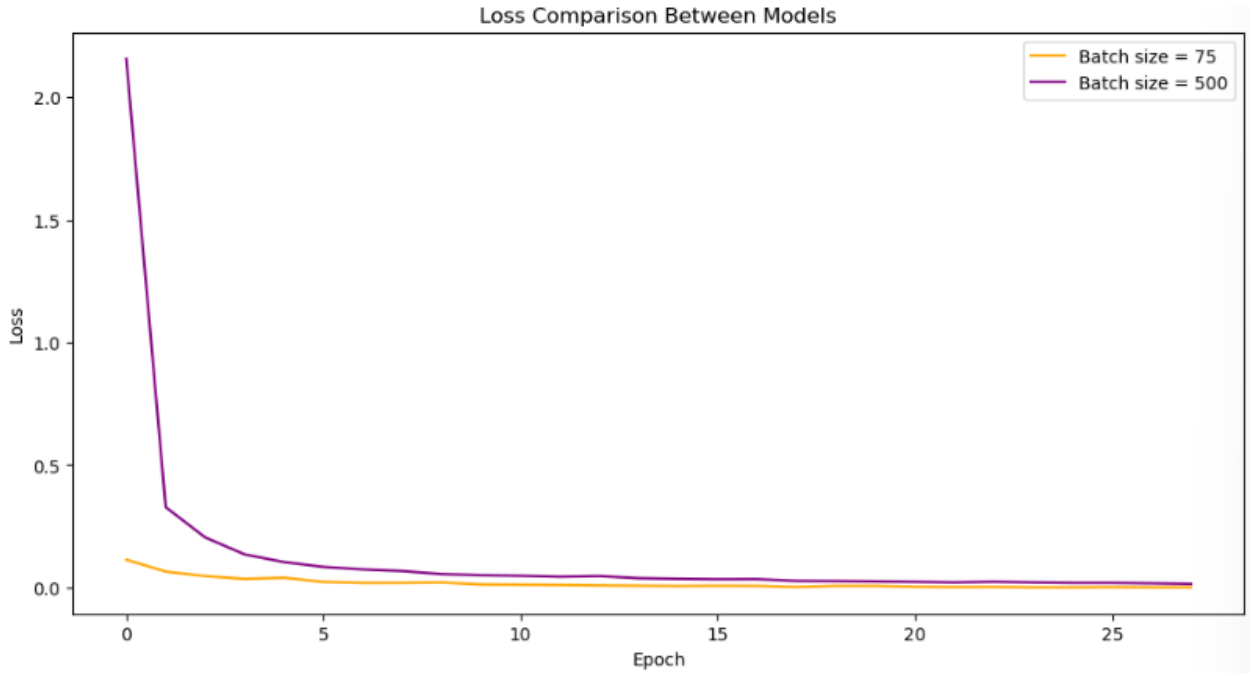
- **Optimizer**: Stochastic Gradient Descent (SGD) with momentum (0.9) and Nesterov acceleration.

- **Learning Rate**: 0.01
- **Loss Function**: Cross-Entropy Loss
- **Batch Sizes**:
    - Experiment 1: 75 samples per batch
    - Experiment 2: 500 samples per batch
- **Number of Epochs**: 28

Training Approaches

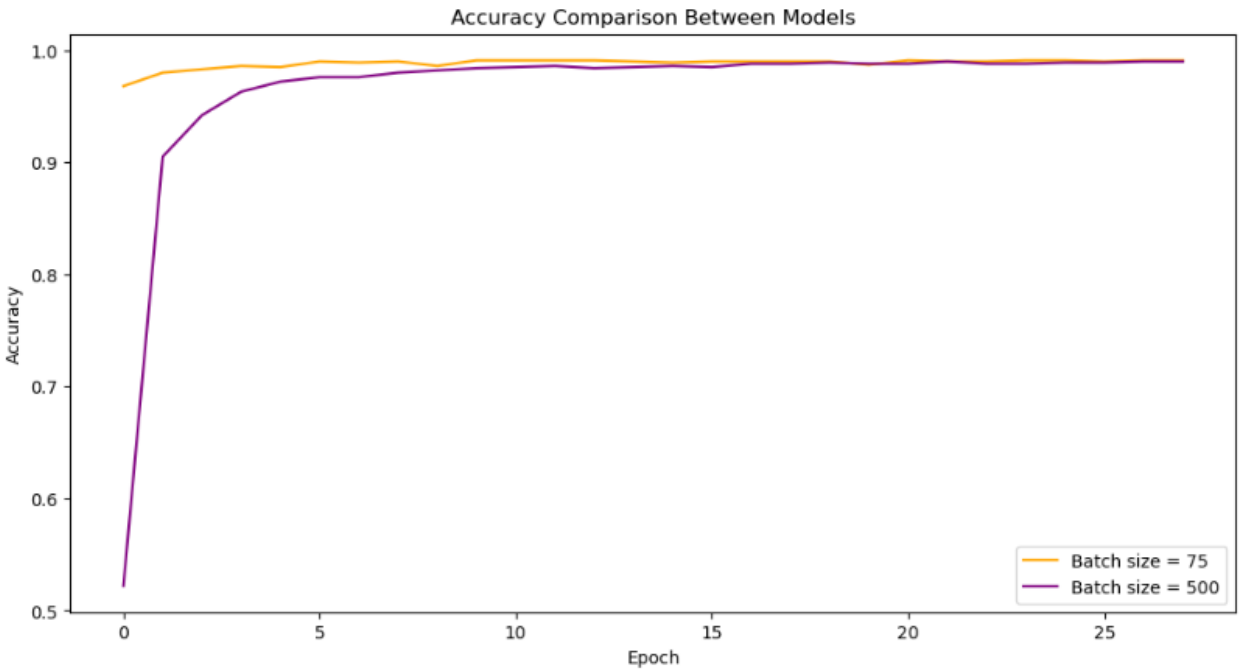
Two models were trained with different batch sizes to observe the impact on performance metrics. After training, the parameters of both models were blended using interpolation to evaluate how the blended models performed on the training and testing datasets.

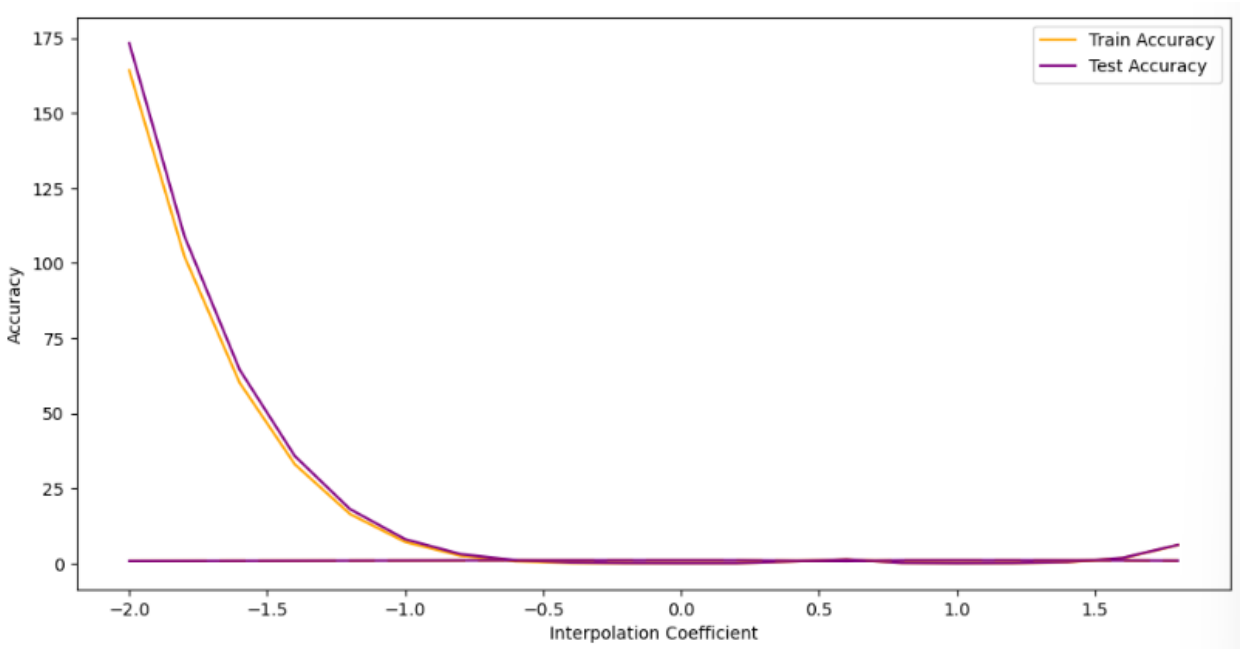
Loss and Accuracy Visualization

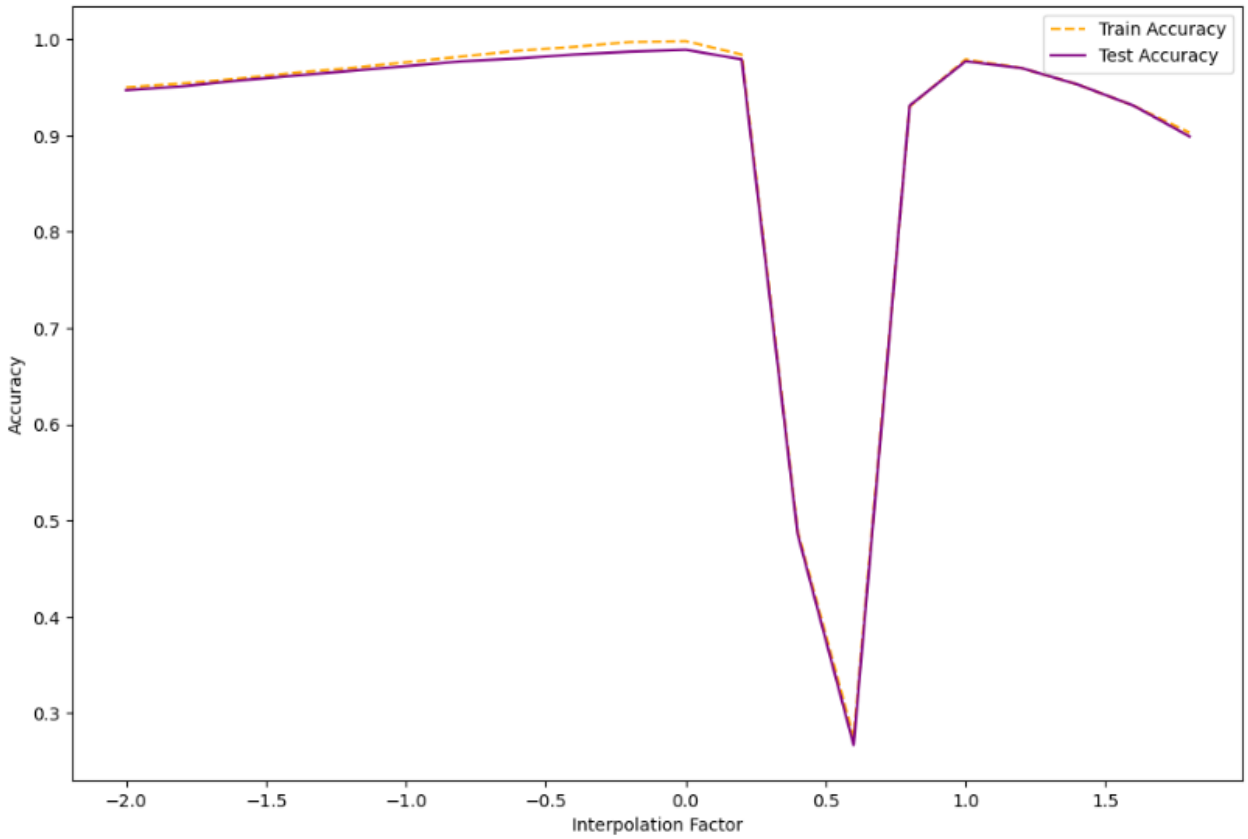


AVINASH GOLLA

DEEP LEARNING ASSIGNMENT 1







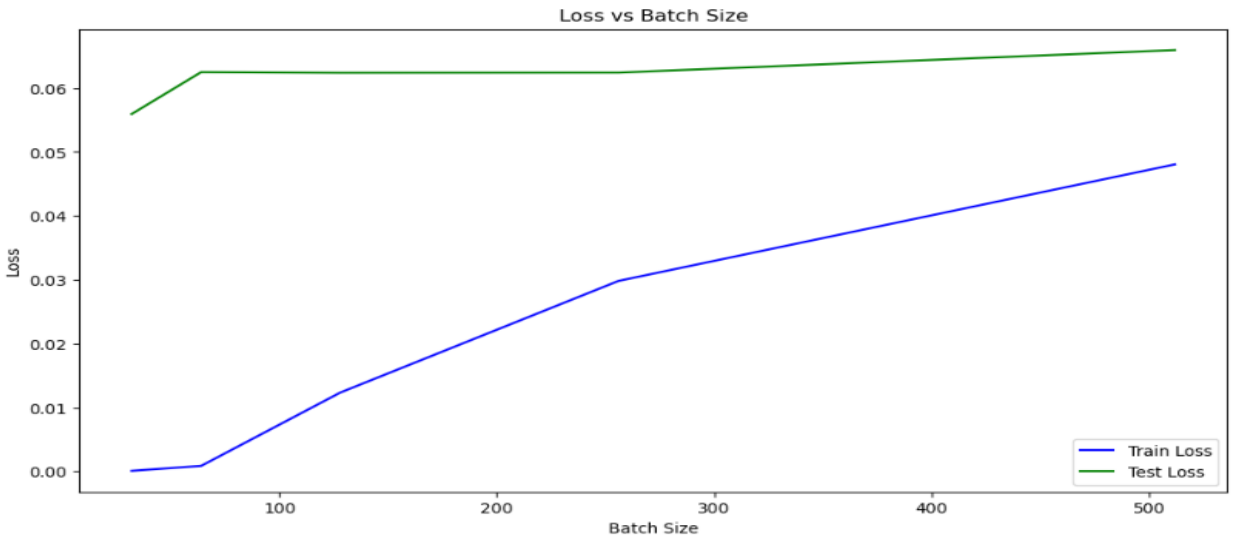
Comments on Results

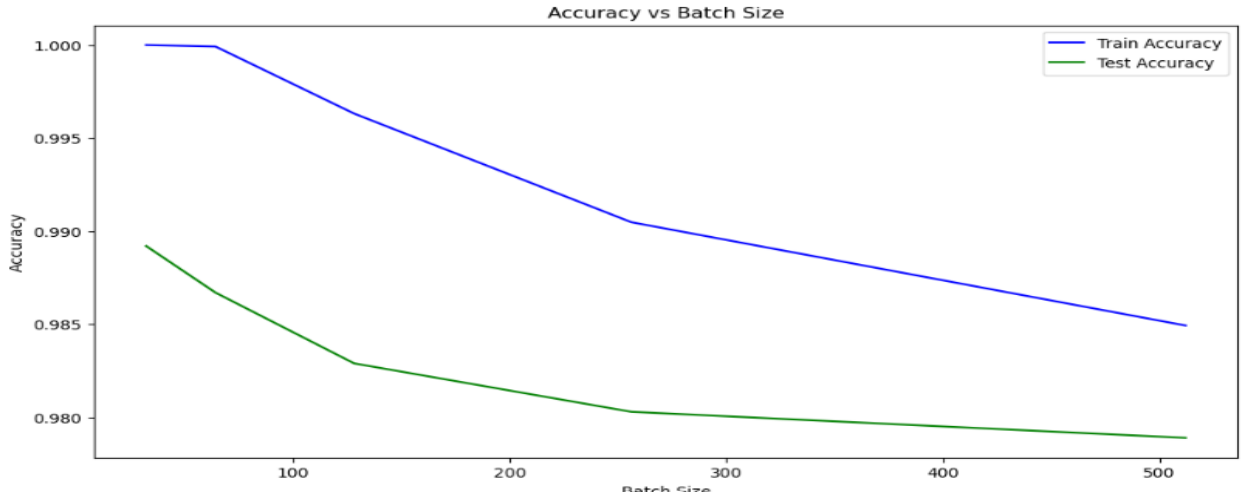
1. **Loss Comparison**: The batch size of 75 typically shows lower training loss than the batch size of 500. This indicates that smaller batch sizes might enable more frequent updates to the model weights, which could result in improved convergence.
2. **Accuracy Comparison**: The accuracy of batch size of 75 is consistently higher than batch size of 500. This reinforces the idea that smaller batch sizes can lead to better generalization on the test set.
3. **Interpolation Results**: This plots of loss and accuracy against the interpolation coefficient reveal interesting insights. As the interpolation coefficient varies, the performance metrics fluctuate, indicating that the blended models can inherit characteristics from both parent models. The optimal performance is observed at certain interpolation values, suggesting that the combination of parameters can yield better results than either model alone.

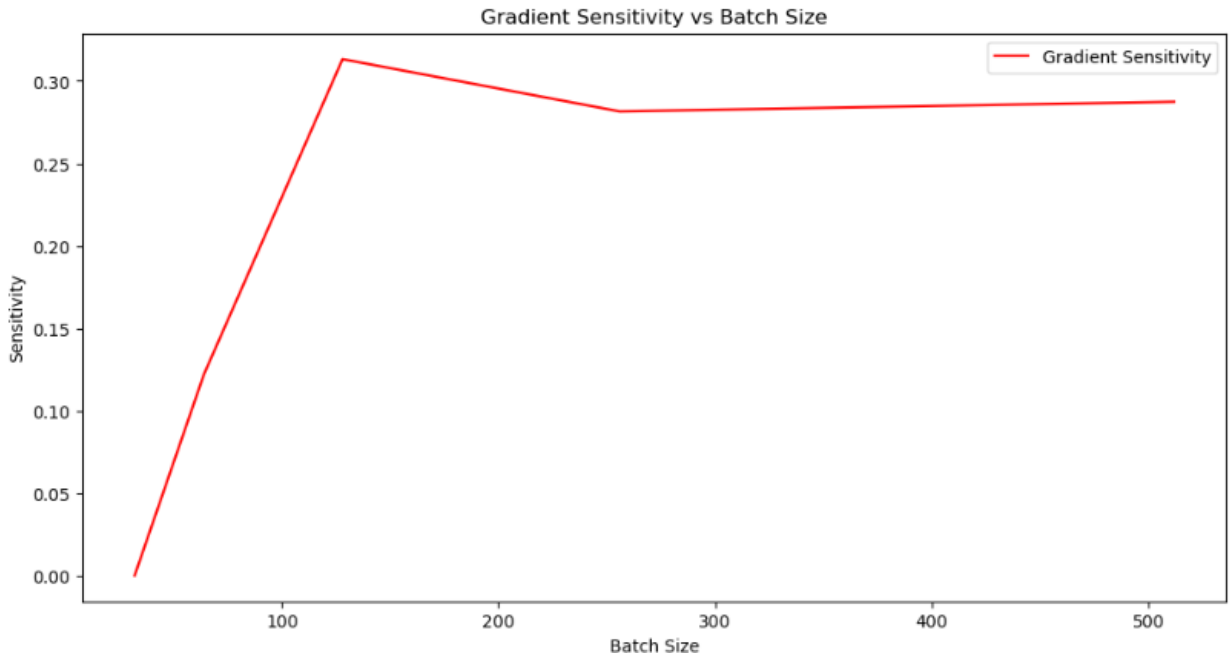
Part2

Training Approach

- The training process is conducted over 50 epochs.
- The loss function used is Cross-Entropy Loss, which is suitable for multi-class classification tasks.
- The experiment assesses the model's performance with a range of batch sizes: 32, 64, 128, 256, and 512.







Comments on Results

- The training and testing loss generally decreases as the batch size increases, indicating that larger batch sizes may help the model converge more effectively.
- The training and testing accuracy improves with larger batch sizes, which may be attributed to the more stable gradient estimates provided by larger batches.
- The gradient sensitivity values show a decreasing trend with increasing batch sizes. This suggests that larger batch sizes may lead to more stable gradients, which can be beneficial for training deep learning models.