CPSC 8430 Deep Learning Homework 1 Report

Name: Aditya Mohan More

CUID: C97857734

Email: more@clemson.edu

Code implementation Github URL:
https://github.com/Skywalker1910/CPSC-8430-Deep-Learning/

HW1: Deep vs Shallow Models

Simulate a function

In this task, I compared the performance of three neural network models of varying complexity when trained to approximate two different functions. The functions being approximated are:

f1(x) = sin(5πx) / 5πx - a continuous function.

f2(x) = sign(sin(5πx)), a non-continuous, function representing a step function.

Models:

We trained three neural networks with varying architectures to evaluate their performance on the given functions:

Model 0 (Deep Model):

Layers: 1 Input, 5 hidden layers with (5, 10, 10, 10, 5 neurons), and 1 output layer.

Total parameters: 401

Model 1 (Medium Complexity Model):

Layers: 1 Input, 4 hidden layers with (10, 18, 15, 4 neurons), and 1 output layer.

Total parameters: 451

Model 2 (Shallow Model):

Layers: 1 Input, 1 hidden layer with 190 neurons, and 1 output layer.

Total parameters: 381

The models were trained using the Adam optimizer with a learning rate of 0.001 for 20,000 epochs.

Training Results:

Training Loss for f1(x)

The following plot shows the training loss (Mean Squared Error) for all three models on the f1(x) function.

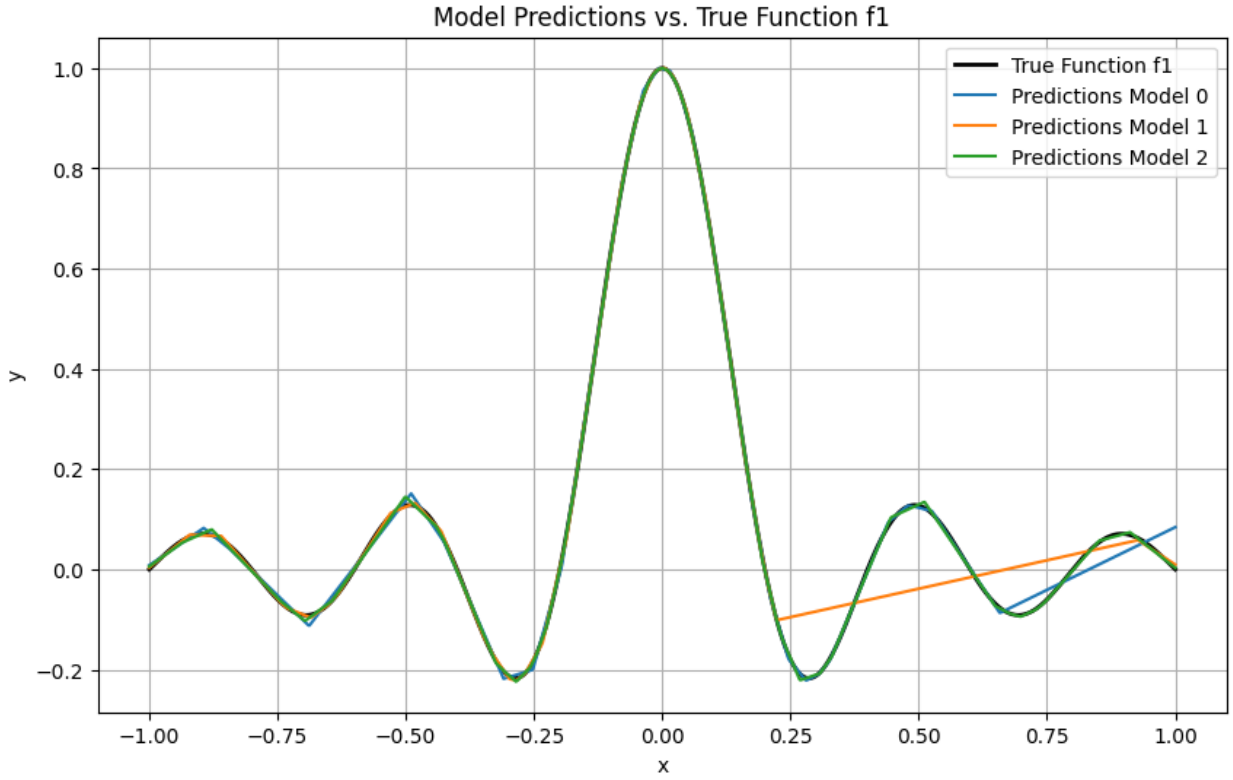
Observation:

All models converge to a very low loss, but Model 0 and Model 1 converge slightly faster.

Model 2, with a single hidden layer, converges but still performs comparably to the deeper models by the end of training.

Predicted vs. Ground Truth for f1(x)

The following plot shows the predicted function curves for each model, compared with the true function f1(x)f_1(x)f1 (x).



Observation:

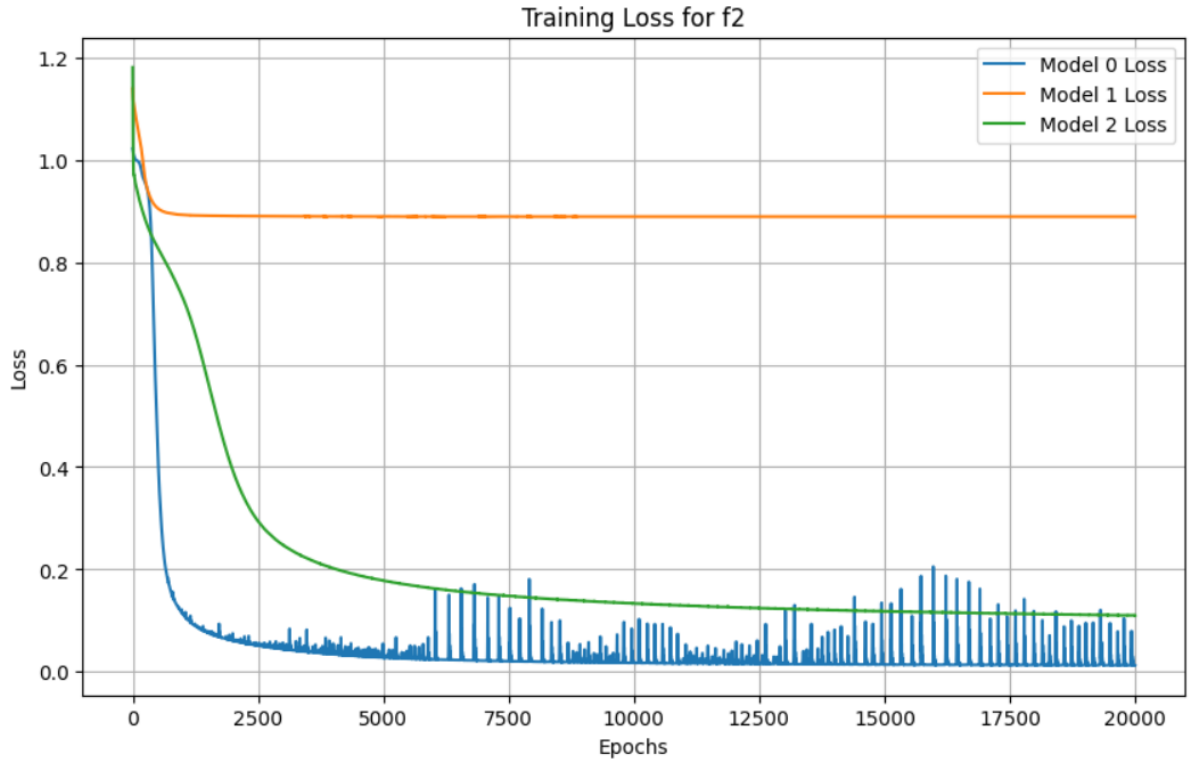
Model 0 and Model 2 show close approximations to the true function curve.

Model 1, on the other hand, struggles to match the oscillatory behavior accurately.

This shows that while shallow models can fit the oscillatory function well (Model 2), having more complex architectures doesn't always guarantee better results.

Training Loss for f2(x)

The following plot shows the training loss for all three models on the f2(x) function.



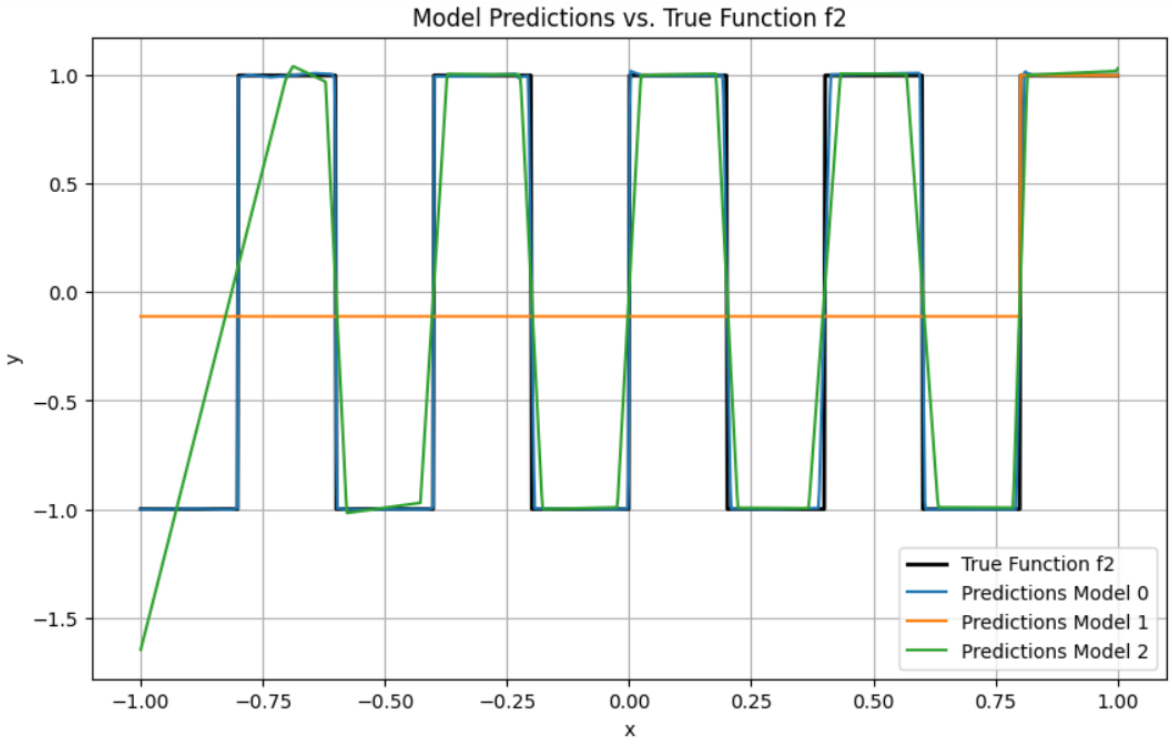
Observation:

Model 0 and Model 2 show clear improvement in loss over time.

Model 1 struggles with the loss plateauing very early, indicating difficulty in learning the non-continuous step function.

Predicted vs. Ground Truth for f2(x)

The following plot shows the predicted function curves for each model, compared with the true function f2(x).



Observation:

Model 0 and Model 2 manage to approximate the step function reasonably well.

Model 1 once again struggles, outputting a nearly flat function that fails to represent the step-like behavior.

Train on actual task

This experiment compares the performance of two Convolutional Neural Networks (CNNs) on the MNIST dataset, a standard benchmark for handwritten digit classification (0-9). We examine how model complexity affects training loss and accuracy.

Dataset

MNIST Dataset:

Training Set: 60,000 images.

Test Set: 10,000 images.

Image Size: 28x28 pixels, grayscale.

Classes: 10 digits (0-9)

Models

Model 0: Shallow CNN:

2 convolutional layers (16 and 32 filters).

1 fully connected layer (128 neurons).

Total Parameters: ~120,000.

Model 1: Deep CNN:

3 convolutional layers (32, 64, and 128 filters).

1 fully connected layer (256 neurons).

Total Parameters: ~500,000.

Both models use ReLU activations, max-pooling, and a final softmax layer for classification.

Training Setup:

Optimizer: Adam.

Learning Rate: 0.001.

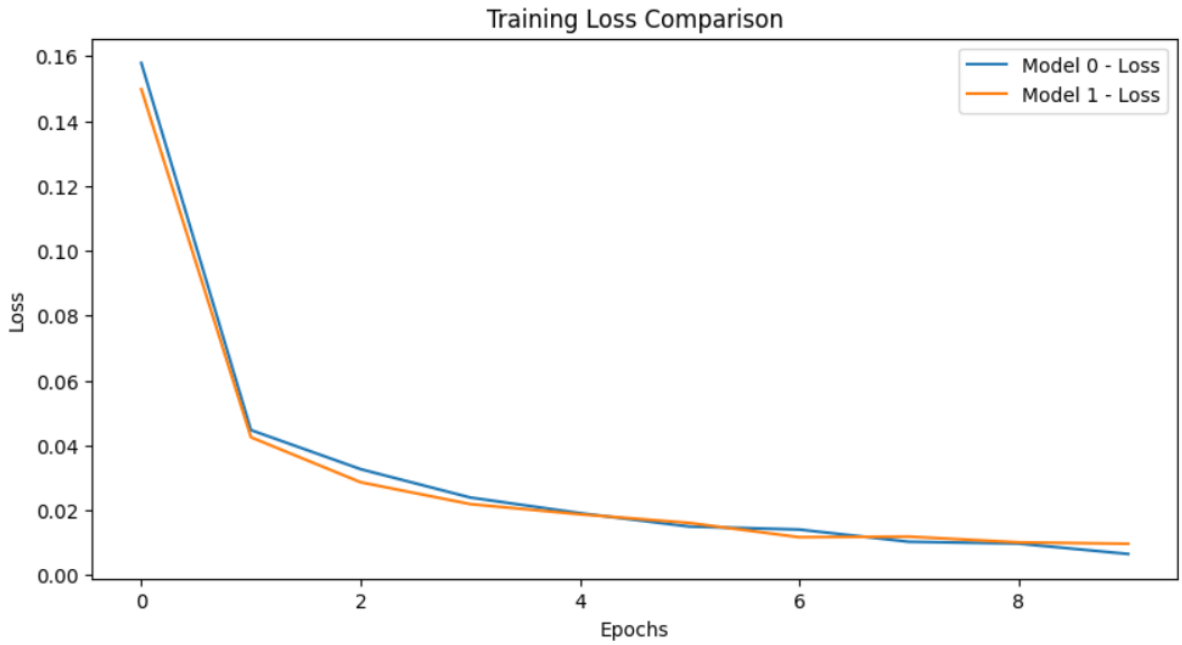
Batch Size: 64.

Epochs: 10.

I trained both models and tracked training loss and accuracy over time.

Results

Training Loss

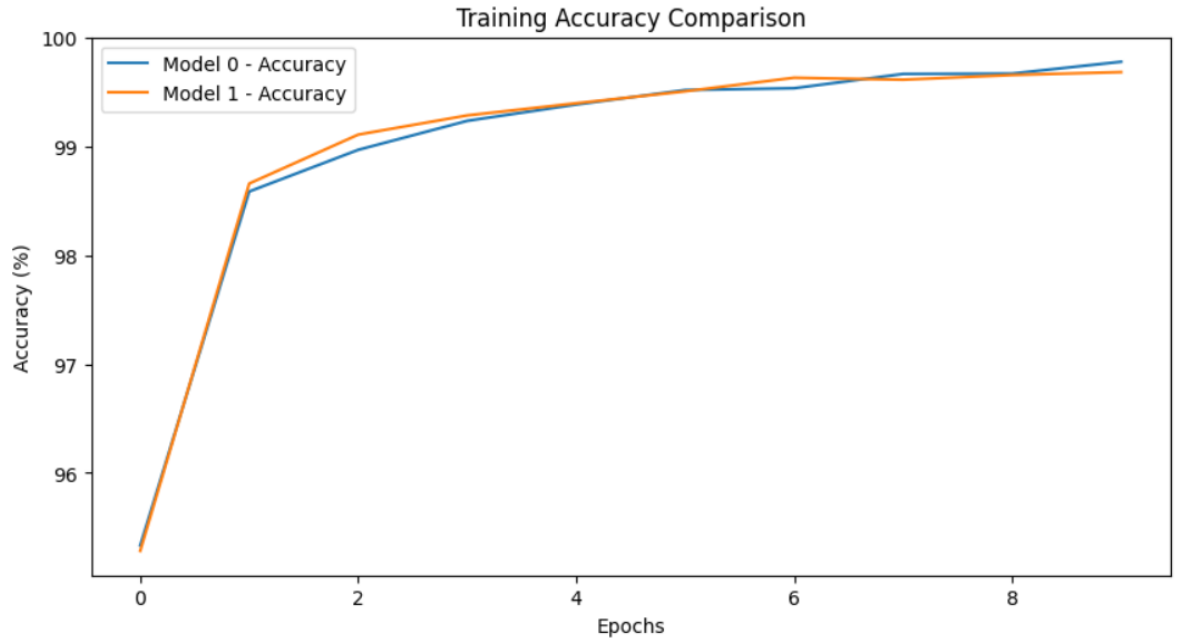


Observation:

Both models reduced loss steadily.

The deep CNN (Model 1) achieved slightly lower loss than the shallow CNN (Model 0).

Training Accuracy



Observation:

Both models achieved high accuracy, with the deep CNN reaching nearly 100% and the shallow CNN close to 99%.

The deeper CNN had slightly better accuracy overall.

HW1-2: Optimization - Visualizing the Optimization Process

I trained a Deep Neural Network (DNN) on the MNIST dataset and visualized the optimization process by collecting and analyzing the model's weights. We reduced the dimensionality of the collected weights using Principal Component Analysis (PCA) to see how they evolved during training and how different training runs compared.

Experiment Settings

Model Architecture:

A simple DNN with three fully connected layers:

Layer 1 (fc1): 128 units.

Layer 2 (fc2): 64 units.

Output Layer: 10 units (classification).

Training Configuration:

Dataset: MNIST (handwritten digits).

Optimizer: SGD (learning rate: 0.01).

Loss Function: Cross-Entropy Loss.

Epochs: 9 (weights collected every 3 epochs).

Training Events: 8 independent training runs.

Data Collection:

Collected weights from Layer 1 (fc1) and the whole model.

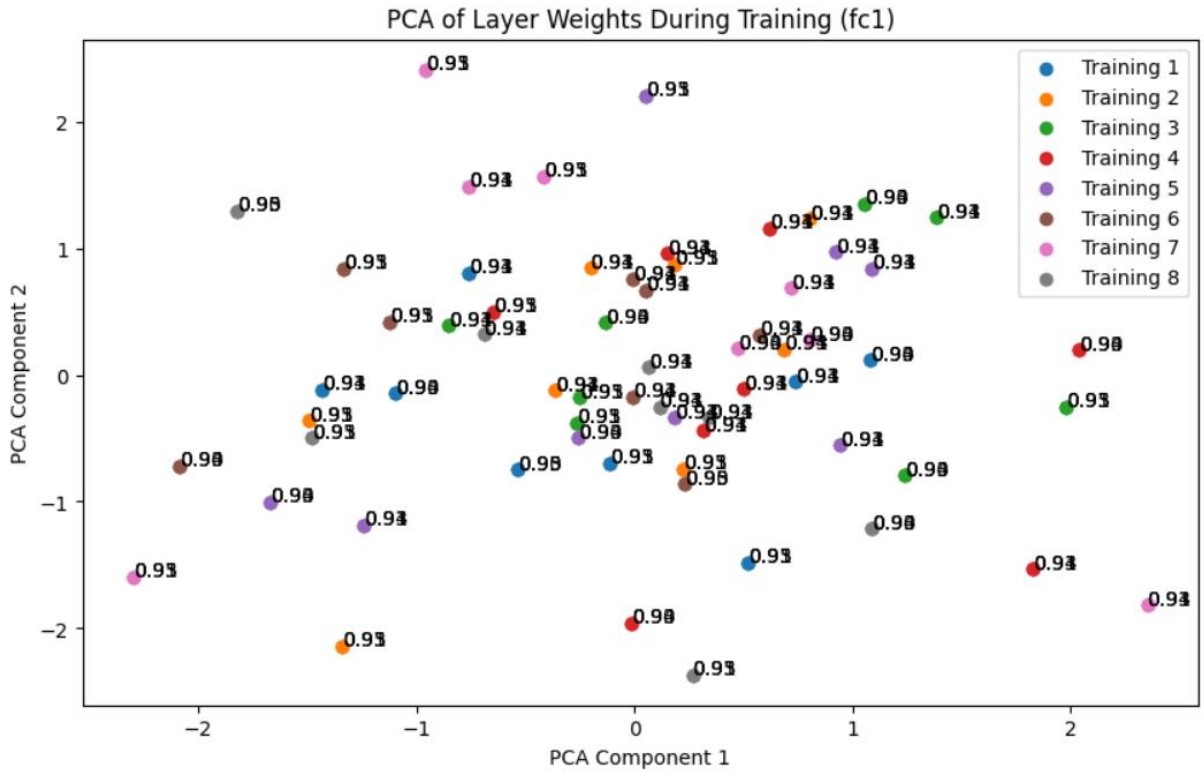
Applied PCA to reduce the weight dimensionality to 2D.

Recorded model accuracy with the collected weights.

Results

Layer 1 Weights Visualization

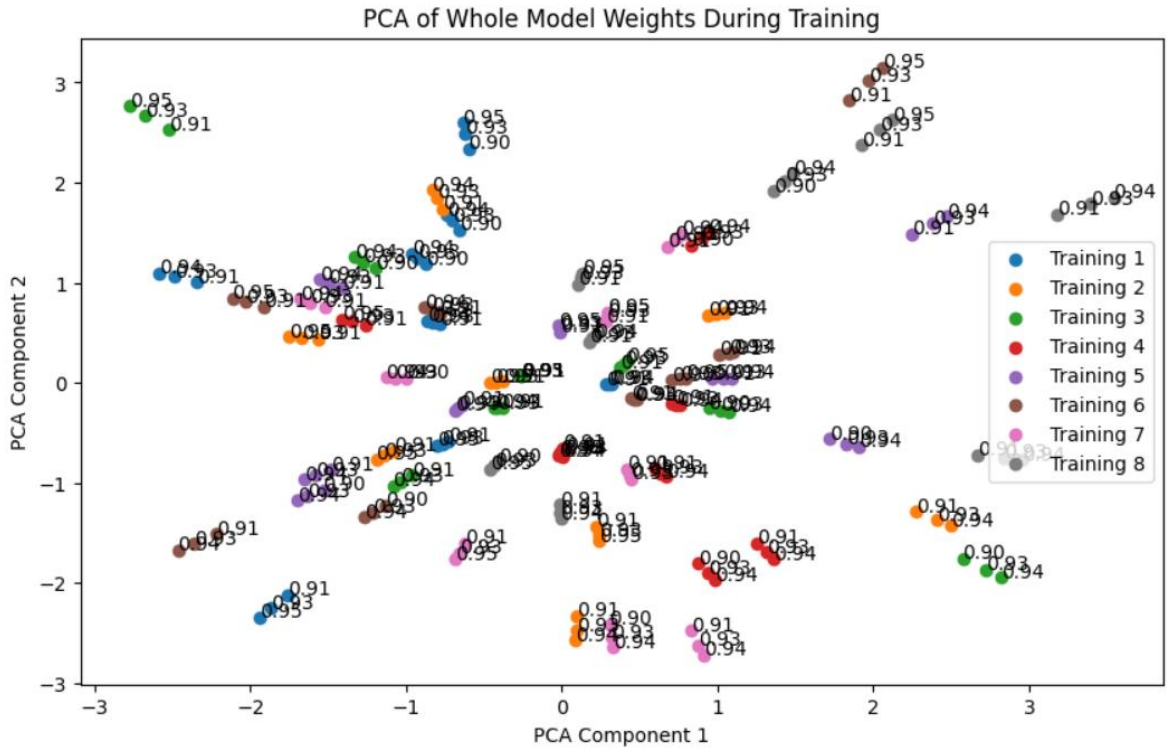
The PCA plot below shows the Layer 1 (fc1) weights across 8 training runs. Points represent weights at different epochs, with accuracy displayed.



Observation: Layer 1 weights follow different paths across training runs but tend to converge toward similar areas, with accuracy reaching up to 99%.

Whole Model Weights Visualization

The PCA plot below shows the whole model weights for all 8 training runs. The accuracy at each epoch is also displayed.



Observation: The whole model weights show more spread initially but also converge toward regions of high accuracy (up to 99%).

HW1-2: Observing Gradient Norm During Training

In this task, I implemented simple NN to examine the behavior of gradient norms during the training of a neural network. Gradients determine how the model's weights are updated, and observing their magnitude helps in understanding the optimization process. Aim is to visualize the relationship between gradient norms and training loss.

Experiment Settings

Dataset: MNIST (handwritten digits).

Model: A simple neural network with three fully connected layers:

Layer 1: 128 units.

Layer 2: 64 units.

Output Layer: 10 units.

Optimizer: SGD (learning rate: 0.01).

Loss Function: Cross-Entropy.

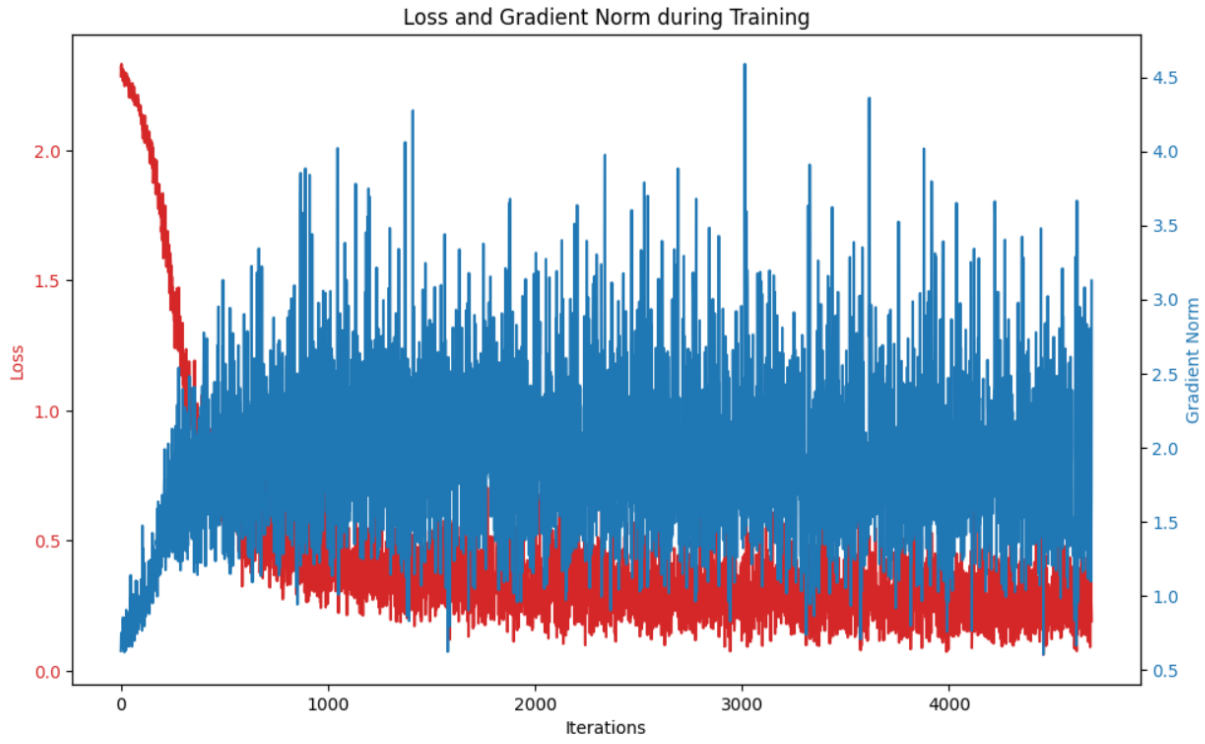
Batch Size: 64.

Epochs: 5.

I recorded the gradient norms and loss at every iteration (batch update).

Results

The plot below shows the gradient norm (blue) and training loss (red) over iterations:



Observations:

Early Stage: Both the gradient norm and loss are high. The model is making large updates to reduce the loss quickly.

Mid Stage: The gradient norm fluctuates, while the loss steadily decreases, indicating that the model is exploring the loss landscape.

Later Stage: The gradient norm decreases, leading to smaller updates as the model converges, and the loss stabilizes.

HW1-3: Generalization - Flatness vs. Generalization

In this subtask, I implemented the relationship between the flatness of the loss landscape and the generalization ability of a neural network. I have trained two models with different settings on the MNIST dataset and interpolated between their weights to analyze how interpolation affects both training and testing performance in terms of loss and accuracy.

Experiment Settings

Dataset:

Training set: 60,000 images.

Testing set: 10,000 images.

Model Architecture

A simple fully connected neural network:

Layer 1: 512 units (ReLU activation).

Output Layer: 10 units (classification).

Training Approaches

Model 1: Trained using the Adam optimizer (learning rate = 0.001) with a batch size of 64.

Model 2: Trained using the Adam optimizer (learning rate = 0.01) with a batch size of 1024.

Interpolation and Evaluation

Interpolation: I interpolated between the weights of the two models using different interpolation ratios (α) ranging from -1 to 2.

Evaluation: For each interpolated model, we evaluated its performance on both the training set and the testing set, measuring:

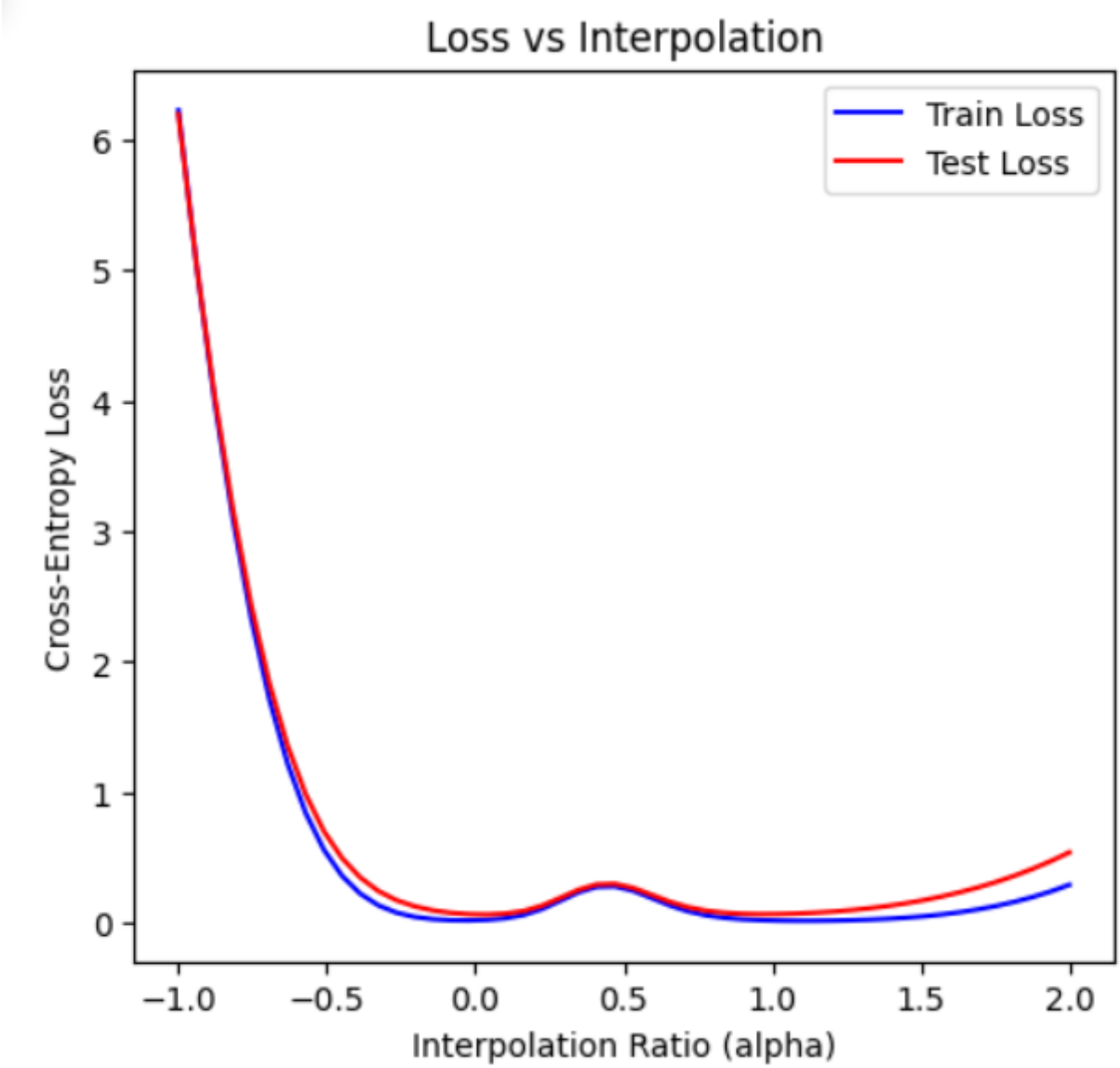
Loss: Cross-entropy loss.

Accuracy: Percentage of correct predictions.

Results

Loss vs. Interpolation Ratio

The plot below shows the training and testing loss as a function of the interpolation ratio:



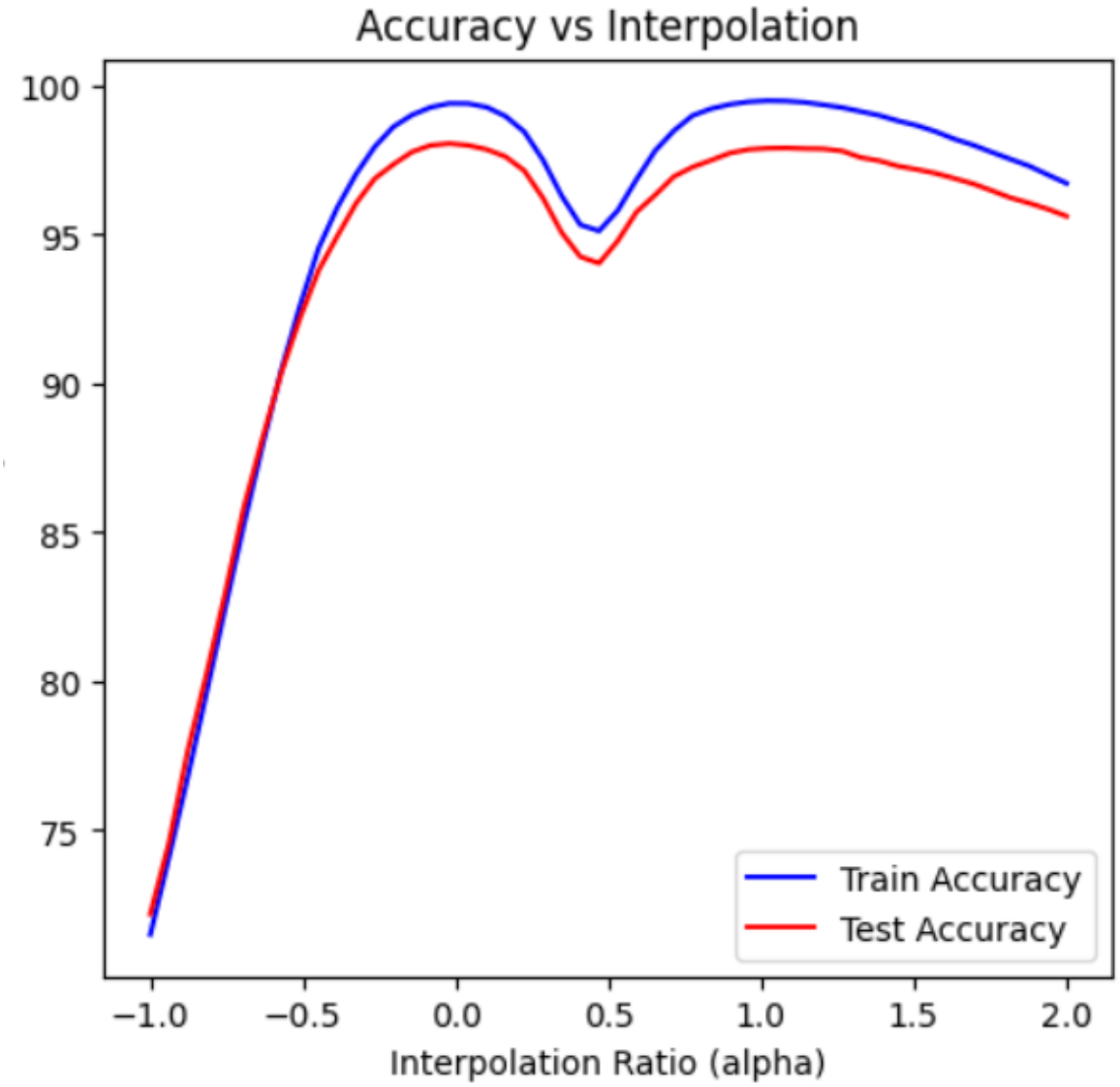
Observation:

The training loss and testing loss both decrease as we move from α = -1 to around α = 0.5.

Beyond α = 1, both losses begin to increase, suggesting that interpolating too far from the original model parameters leads to worse performance.

Accuracy vs. Interpolation Ratio

The following plot shows the training and testing accuracy as a function of the interpolation ratio:



Observation:

Training accuracy peaks around α = 0.5 and decreases after α = 1.

Testing accuracy follows a similar trend, with the best performance occurring around α = 0.5, confirming that this interpolation ratio provides a balance between good training and generalization performance.

HW1-3: Generalization - Number of Parameters vs. Generalization

This experiment investigates how the number of model parameters affects training and testing performance. I have trained 10 Convolutional Neural Networks (CNNs) of

increasing size on the MNIST dataset and analyzed how the number of parameters impacts both training and testing loss and accuracy.

Implementation Settings

Task: MNIST digit classification.

Dataset:

Training Set: 60,000 images.

Test Set: 10,000 images.

Model Architectures

I trained 10 different CNN models, from simple to complex. The models had varying numbers of convolutional layers, filters, and fully connected layers. The smallest model (CNN1) had around 18,000 parameters, while the largest model (CNN10) had over 10 million.

Training Configuration

Optimizer: Adam (learning rate = 0.001).

Loss Function: Cross-Entropy.

Batch Size: 128.

Epochs: 10.

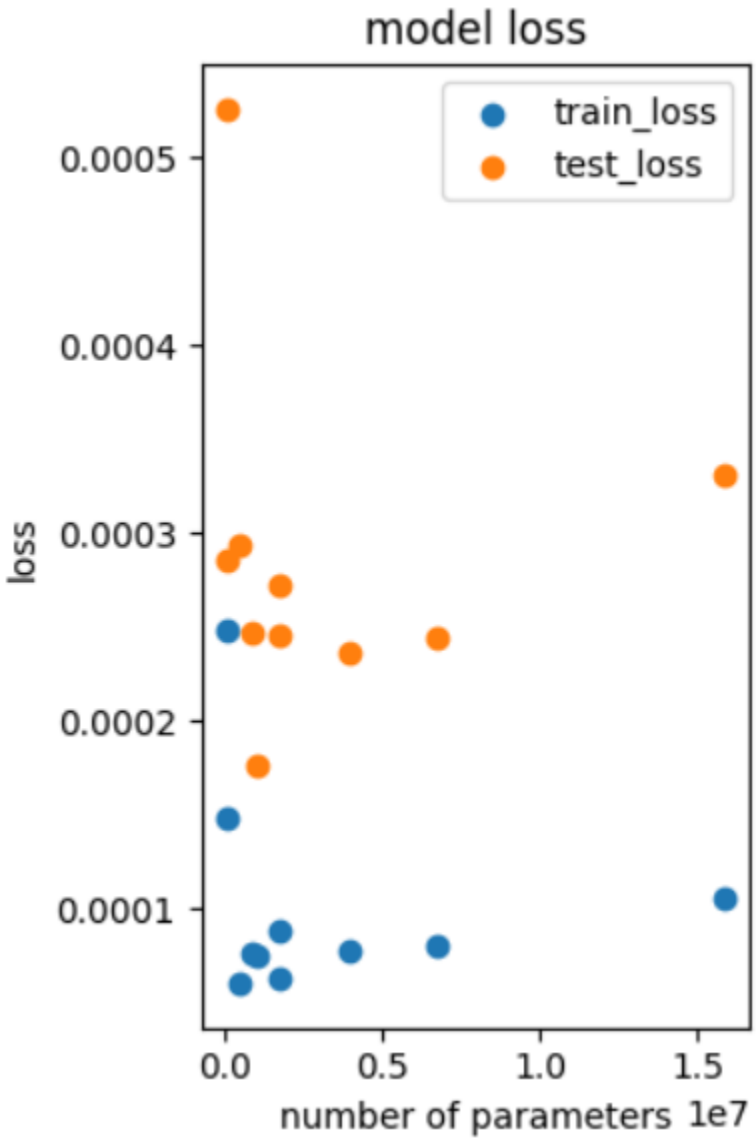
Data Collected

Training Loss/Accuracy and Testing Loss/Accuracy were recorded for each model.

Number of Parameters for each model was recorded to examine the relationship with performance.

Results

Loss vs. Number of Parameters

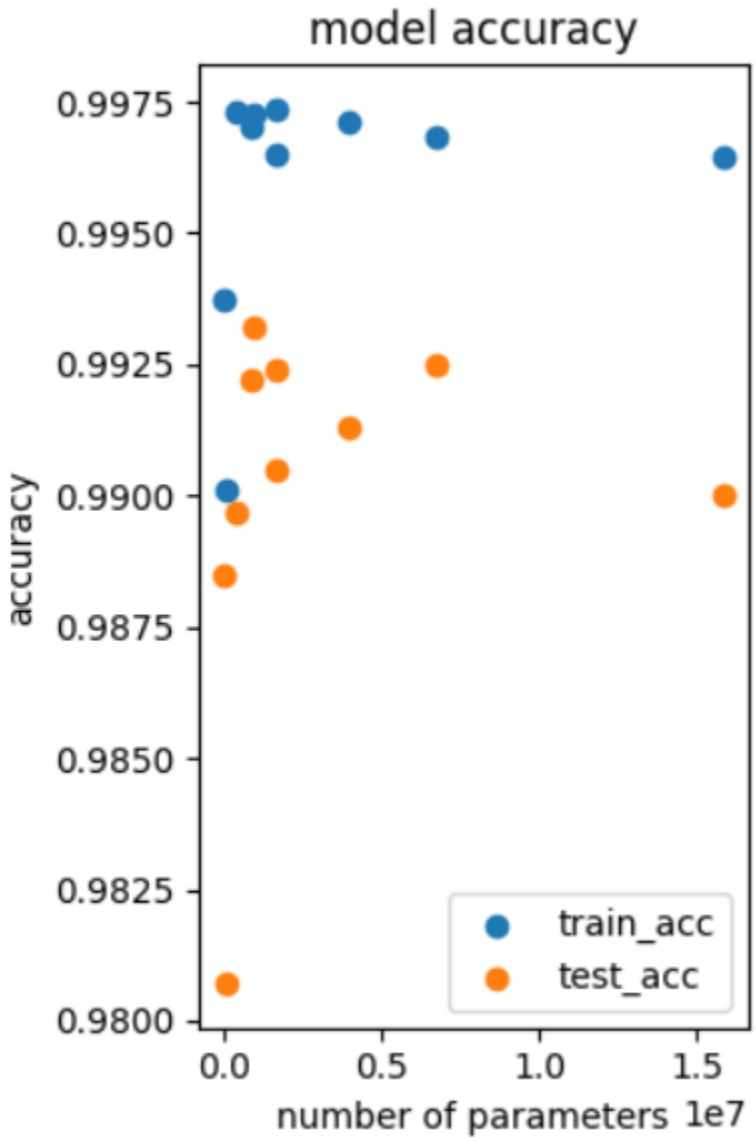


Observation:

Training loss decreases as model size increases, showing that larger models fit the training data better.

Testing loss initially decreases with larger models but starts increasing beyond a certain size, indicating overfitting.

Accuracy vs. Number of Parameters



Observation:

Training accuracy improves with larger models, and the largest models achieve nearly perfect accuracy on the training data.

Testing accuracy increases initially but plateaus or decreases for the largest models, indicating that very large models may overfit.

HW1-3: Can a Neural Network Fit Random Labels?

The aim of this task is to see if a neural network can learn from random labels using the MNIST dataset. We are testing the network's ability to memorize noise and how it performs on unseen data.

Implementation Setting

Dataset: MNIST (60,000 training images, 10,000 testing images).

Data Preprocessing:

Images resized from 28×2828 \times 2828×28 to 32×3232 \times 3232×32.

Normalized to range [0, 1].

Random Labeling:

Training labels were randomly replaced with values between 0 and 9.

Model Architecture

A simple convolutional neural network (CNN) was used:

Convolutional Layers:

Two layers with filters of size 6×56 \times 56×5 and 16×516 \times 516×5.

Fully Connected Layers:

Two hidden layers (120 and 84 neurons) and a final output layer of 10 neurons.

Activation: ReLU was applied after each layer.

Pooling: Max pooling was used to reduce feature map size.

Training Settings

Loss Function: Cross-entropy loss.

Optimizer: Stochastic Gradient Descent (SGD) with learning rate 0.01.

Batch Size:

Training: 64

Testing: 1000

Epochs: 10

Results

Loss and Accuracy over Epochs

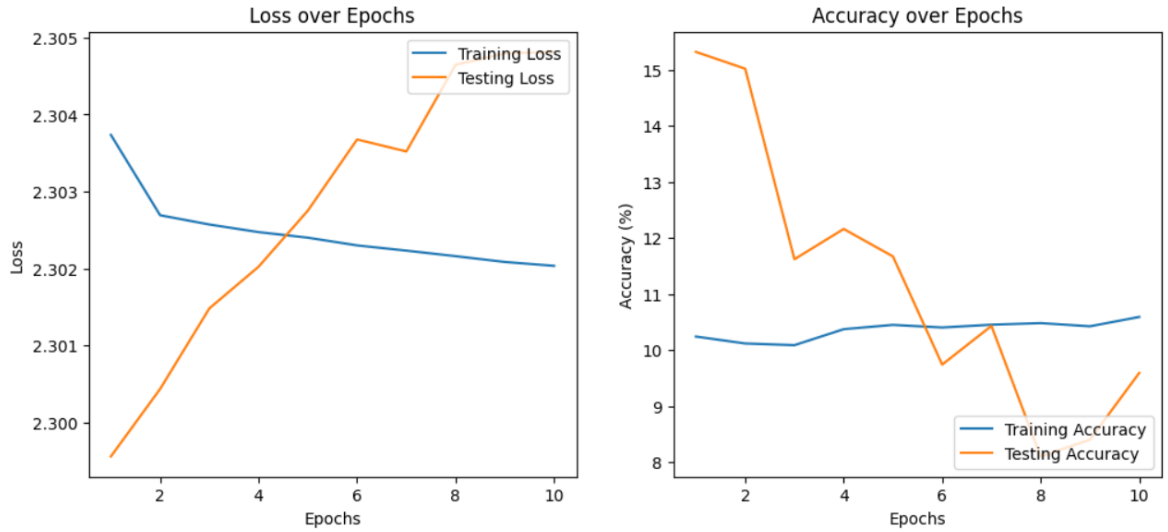


Figure: Loss and Accuracy over Epochs

Training Loss: The model's training loss decreased over time, indicating that the network was able to memorize the random labels.

Testing Loss: The testing loss fluctuated, showing the network could not generalize to the correct test labels.

Training Accuracy: Training accuracy stayed low (~10-11%), meaning the model struggled to memorize the random labels.

Testing Accuracy: Testing accuracy stayed around 10%, equivalent to random guessing, showing that the model didn't generalize.