CPSC 8430

Fall 2024

Homework 1

Abdullah Al Mamun

GitHub repository: https://github.com/abdullm/CPSC_8430_Fall24_AM_HW1.git

1. HW1-1

1.1. Simulate a Function

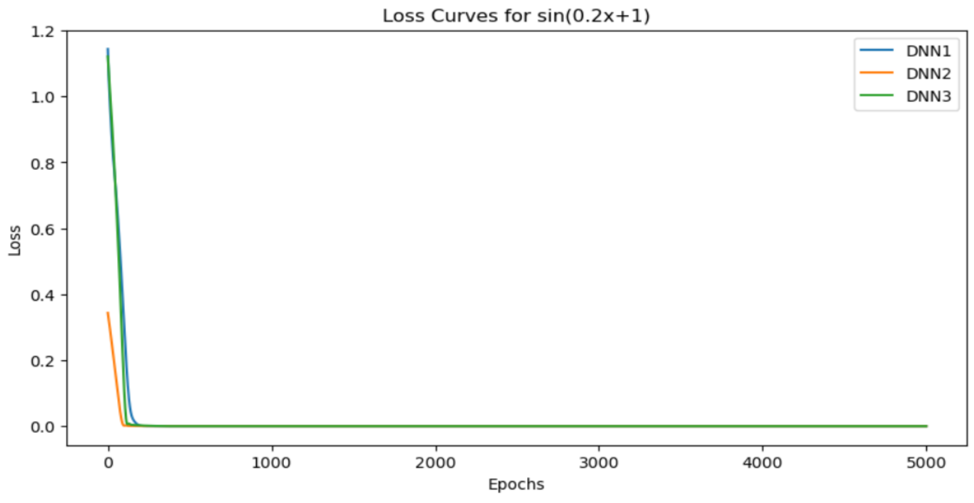
Models Used

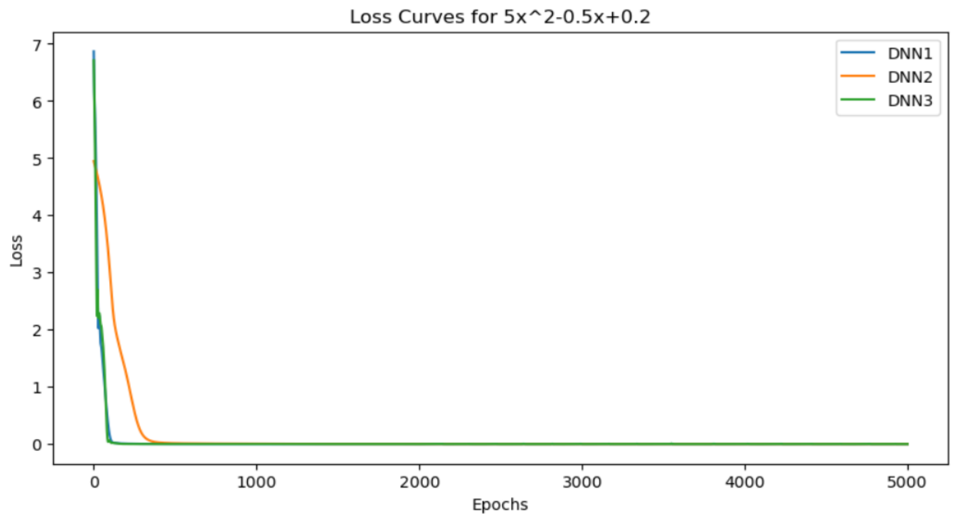
- Three Deep Neural Network (DNN) models have been used:
    1. DNN1 with 4 layers (number of parameters = 566)
    2. DNN2 with 6 layers (number of parameters = 561)
    3. DNN3 with 8 layers (number of parameters = 568)
- Hyperparameter tuning was performed in terms of learning rate (0.0001, 0.001, and 0.01) and optimizer (Adam, SGD, RMSprop)
- Number of epochs = 5000
- Loss function used: MSE

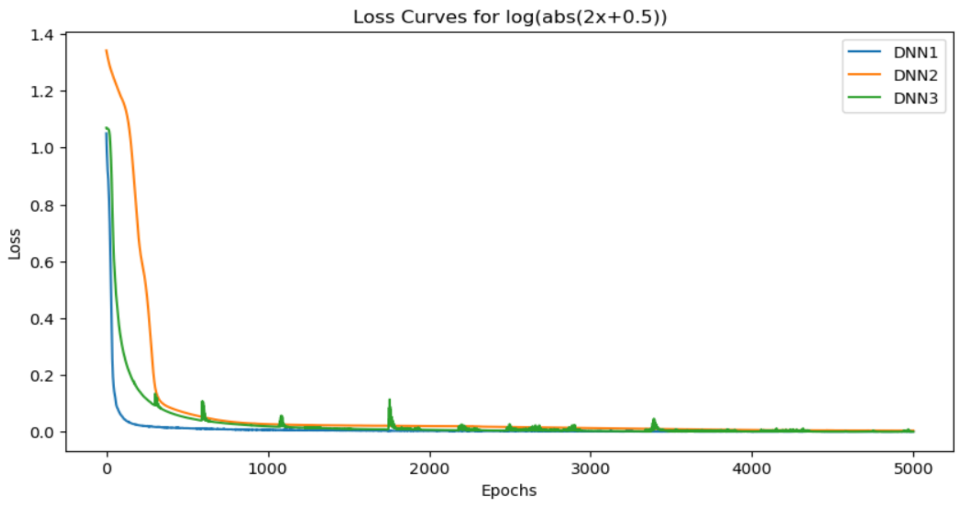
Functions Used

- $\sin(0.2x + 1)$
- $5x^2 - 0.5x + 0.2$
- $\log(|2x + 0.5|)$

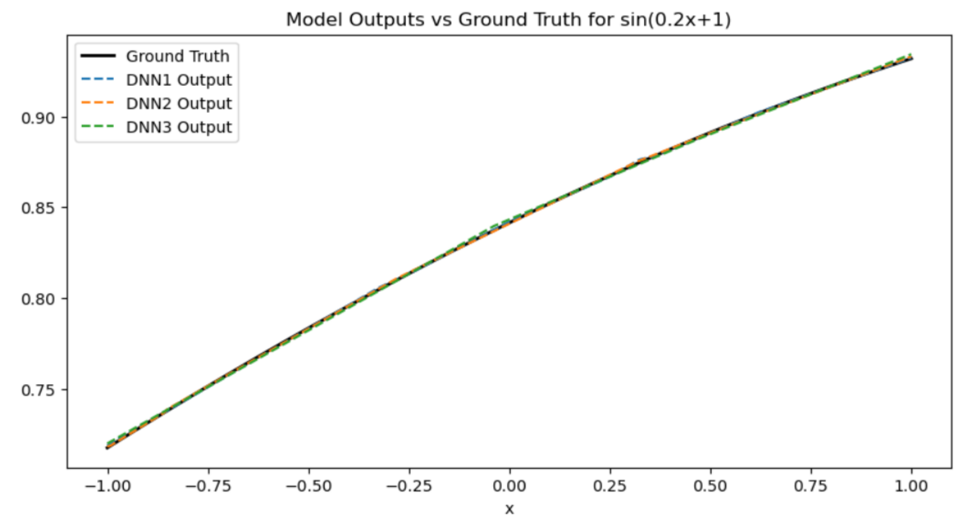
Training loss plots

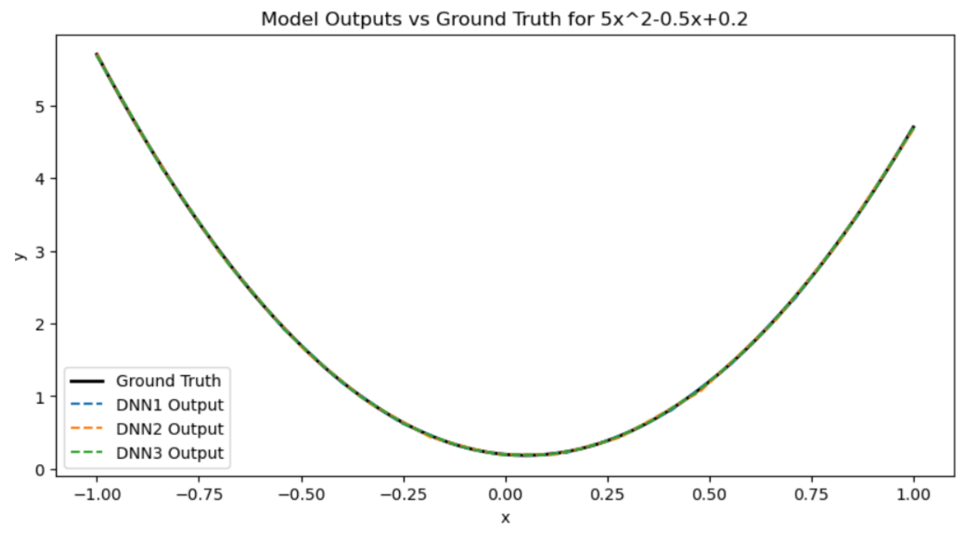


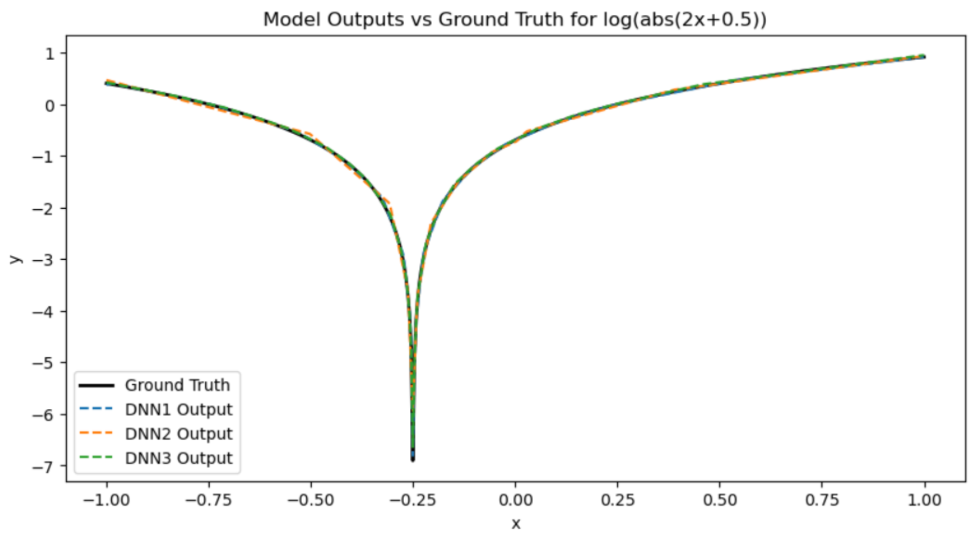




Model outputs vs. ground truth plots







Comments on results

The results from the above two plots show that the three models (DNN1, DNN2, and DNN3) converge similarly in their loss values for all three functions: sin(0.2x+1), 5x^2 - 0.5x + 0.2, and log(abs(2x+0.5)). Despite initial differences in how rapidly each model reduces the loss, they all eventually stabilize to low loss values (the log function taking more epochs to stabilize compares to sine and second-order polynomial functions. The close alignment of the model outputs with the ground truth suggests that all three models are capable of generalizing well to these functions.

1.2. Train on actual tasks

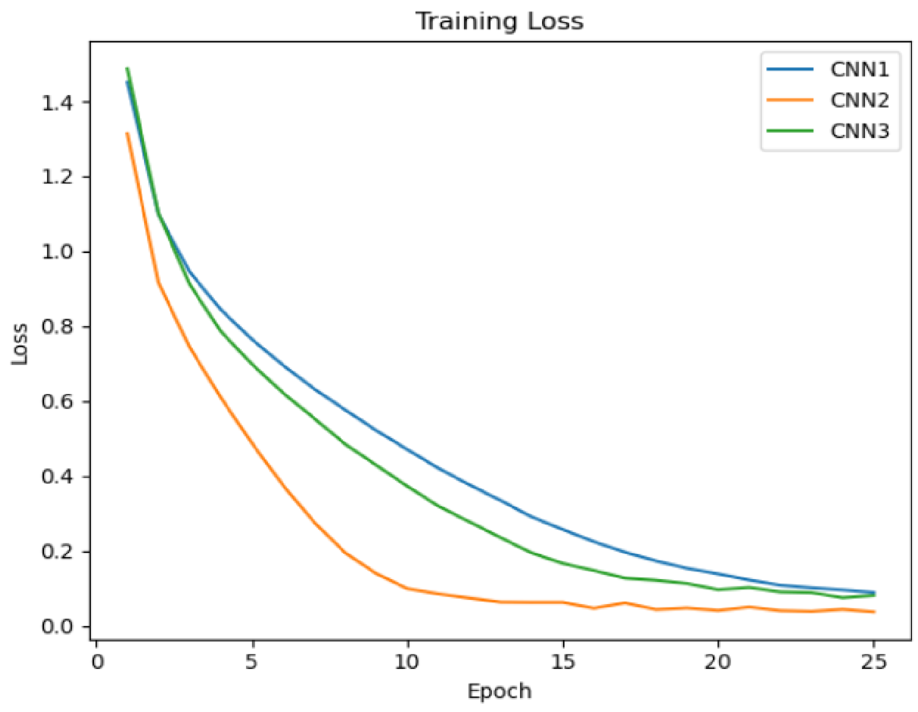
Models Used

- Three Convolutional Neural Network (CNN) models have been used:
    1. CNN1 with 2 convolutional layers, 1 pooling layer, and 2 fully connected layers (number of parameters = 268650)
    2. CNN2 with 2 convolutional layers, 1 pooling layer, and 2 fully connected layers (number of parameters = 1070794)
    3. CNN3 with 3 convolutional layers, 1 pooling layer, and 2 fully connected layers (number of parameters = 288554)
- Learning rate used: 0.001
- Optimizer used: Adam
- Loss function used: Cross-entropy
- Number of epochs: 25
- Batch size: 64

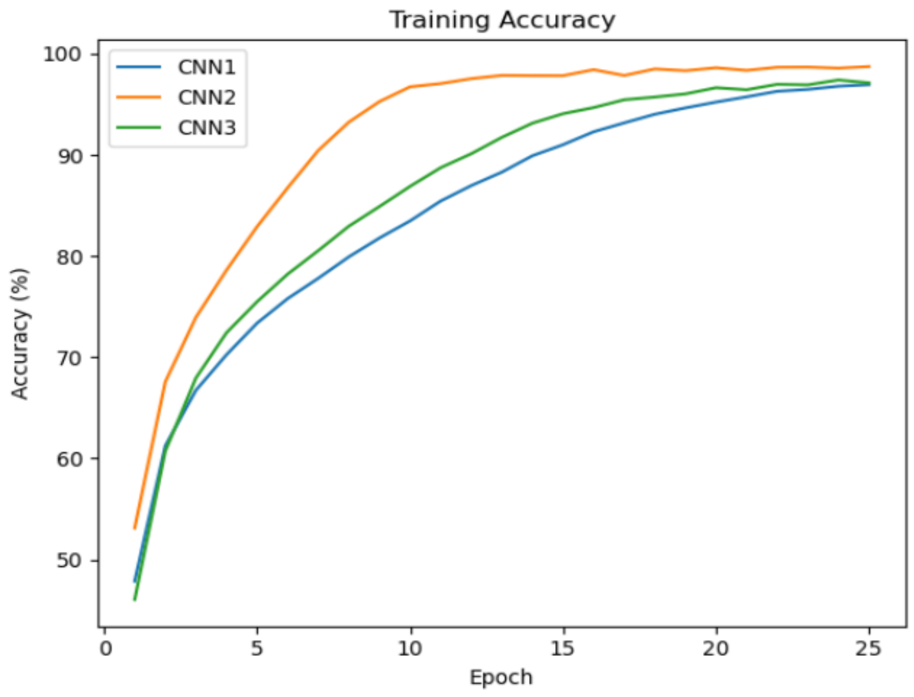
Task Used

- CIFAR-10

Training loss plot



Training accuracy plot



Comments on results

The training loss and accuracy plots show that all three CNN models (CNN1, CNN2, and CNN3) successfully reduced the training loss and increased accuracy as the number of epochs progressed. CNN2, with the highest number of parameters (1,070,794), achieved the fastest convergence and the highest accuracy, while CNN1 with the least number of parameters (268,650) had slower improvement. This indicates that models with more parameters tend to have better capacity to fit the training data; however, CNN3, with fewer parameters than CNN2, still achieved similar performance.

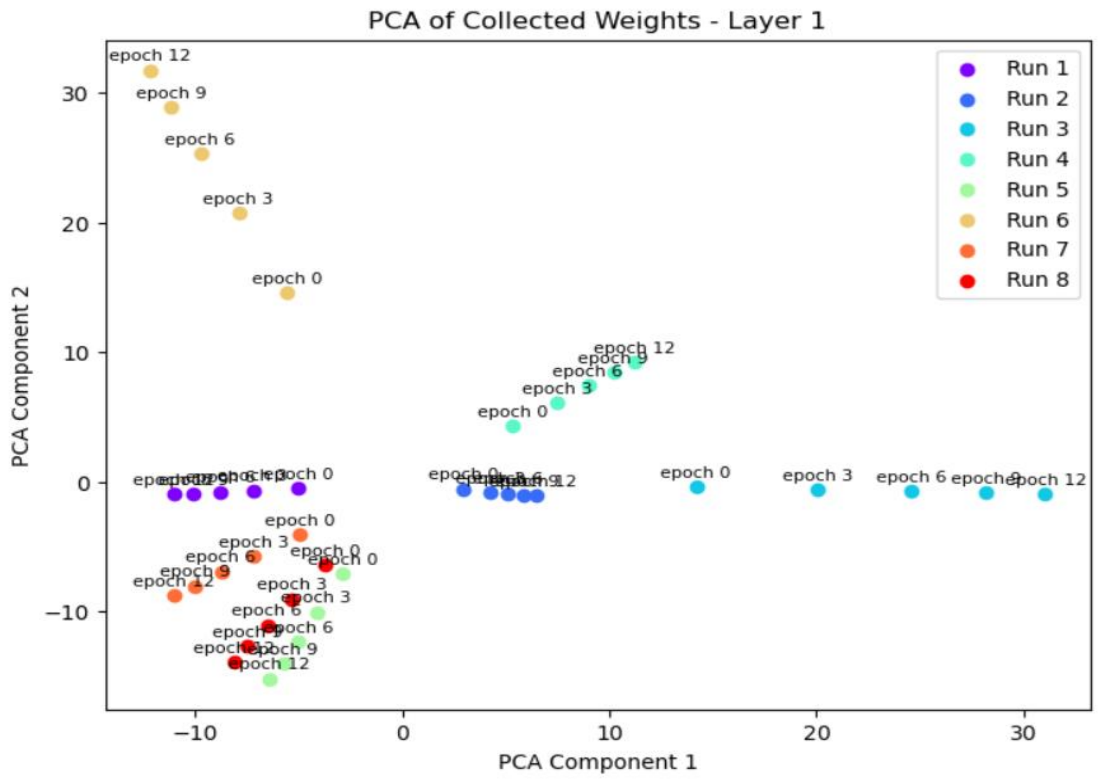
2. HW1-2

2.1. Visualize the optimization process

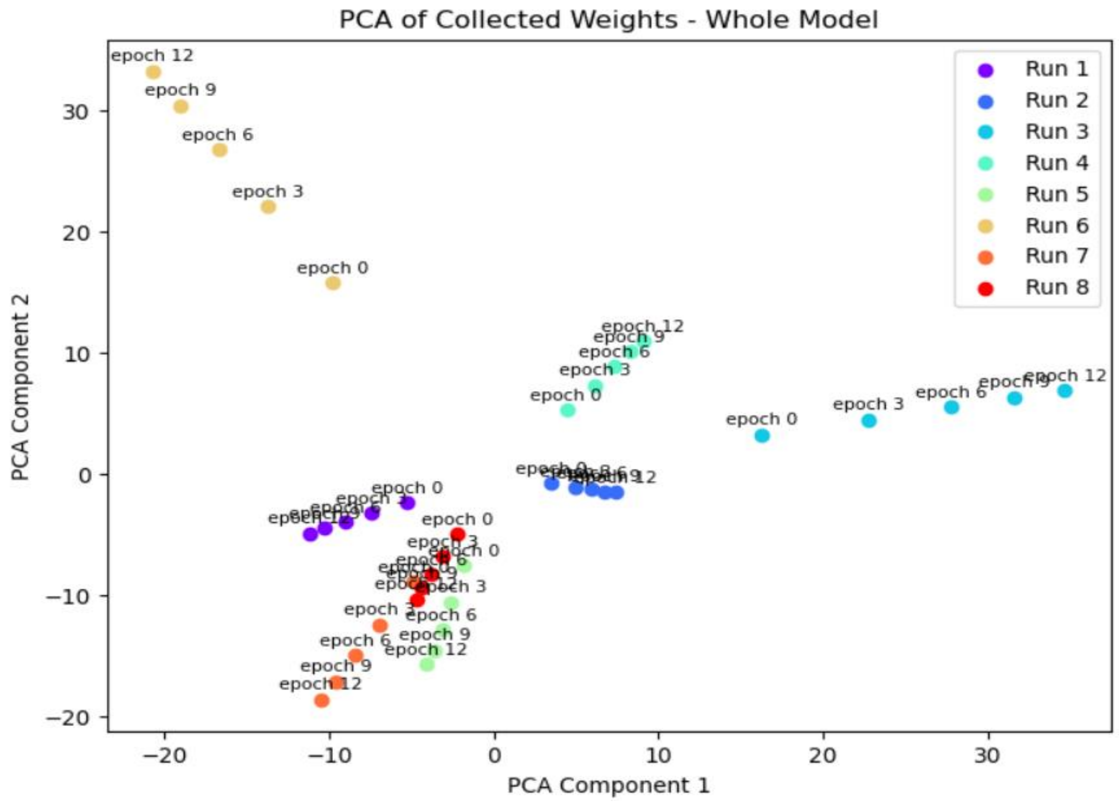
Experiment settings

- Model used: a DNN with 3 layers
- Loss function used: Cross-entropy
- Task used: MNIST
- Learning rate used: 0.001
- Number of epochs: 15
- Optimizer used: Adam
- Batch size: 64
- Cycle to record model parameters: in every 2 epochs during training
- Parameter collections: Whole model weights and layer 1 (fc1) weights were collected in every 3 epochs
- PCA for dimension reduction: Weights were reduced to 2 principal components
- Training runs: The experiment is repeated over 8 independent timing runs

PCA of collected weights (layer 1)



PCA of collected weights (whole model)



Comments on results

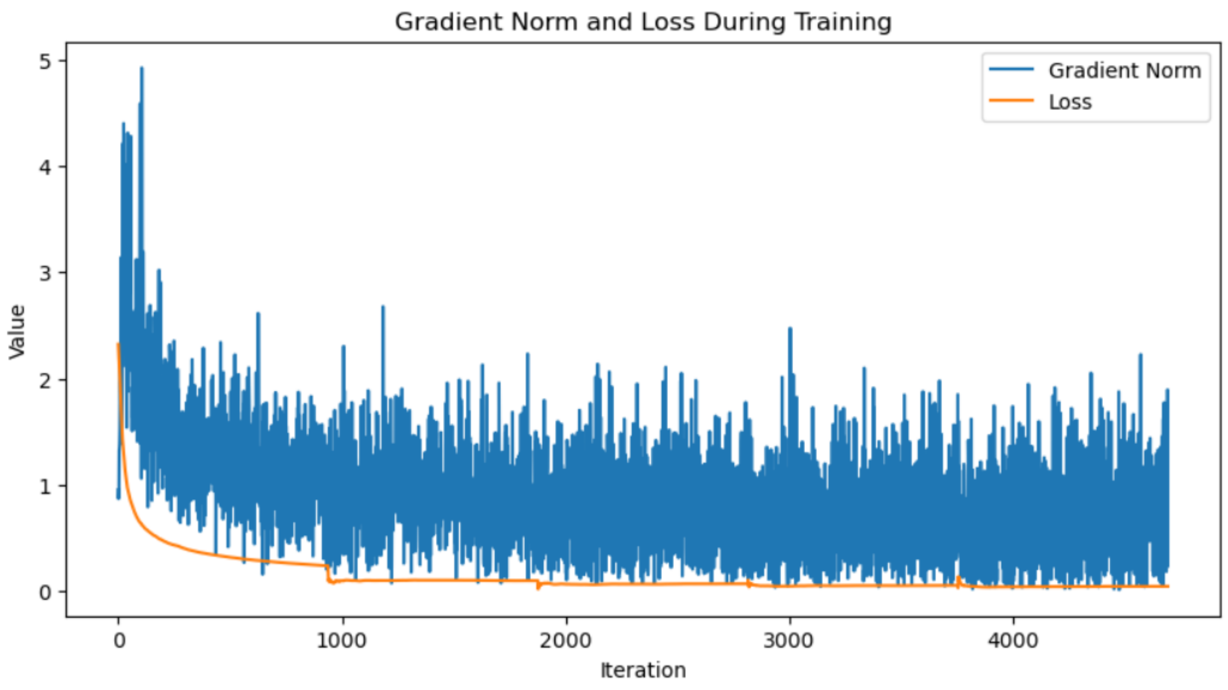
The PCA plots that includes the weights across different epochs show distinct trajectories in the PCA space. Each run begins from a different starting point (epoch 0), and over time (epochs 3, 6, 9, and 12), the weights shift, indicating learning and optimization as the model adjusts its parameters. Interestingly, weights from some runs converge toward specific areas in the PCA space, which could imply similar solutions despite different starting points or paths during training.

In addition, the spread in the PCA space for the whole model appears larger than for layer 1, which implies more diversity in learning across all layers compared to layer 1 alone.

2.2. Observe gradient norm during training

Model/training parameters

- Model used: a DNN with 3 layers
- Loss function used: Cross-entropy
- Task used: MNIST
- Learning rate used: 0.001
- Number of epochs: 5
- Iteration per epoch: 1000
- Optimizer used: Adam
- Batch size: 64



Comments on results

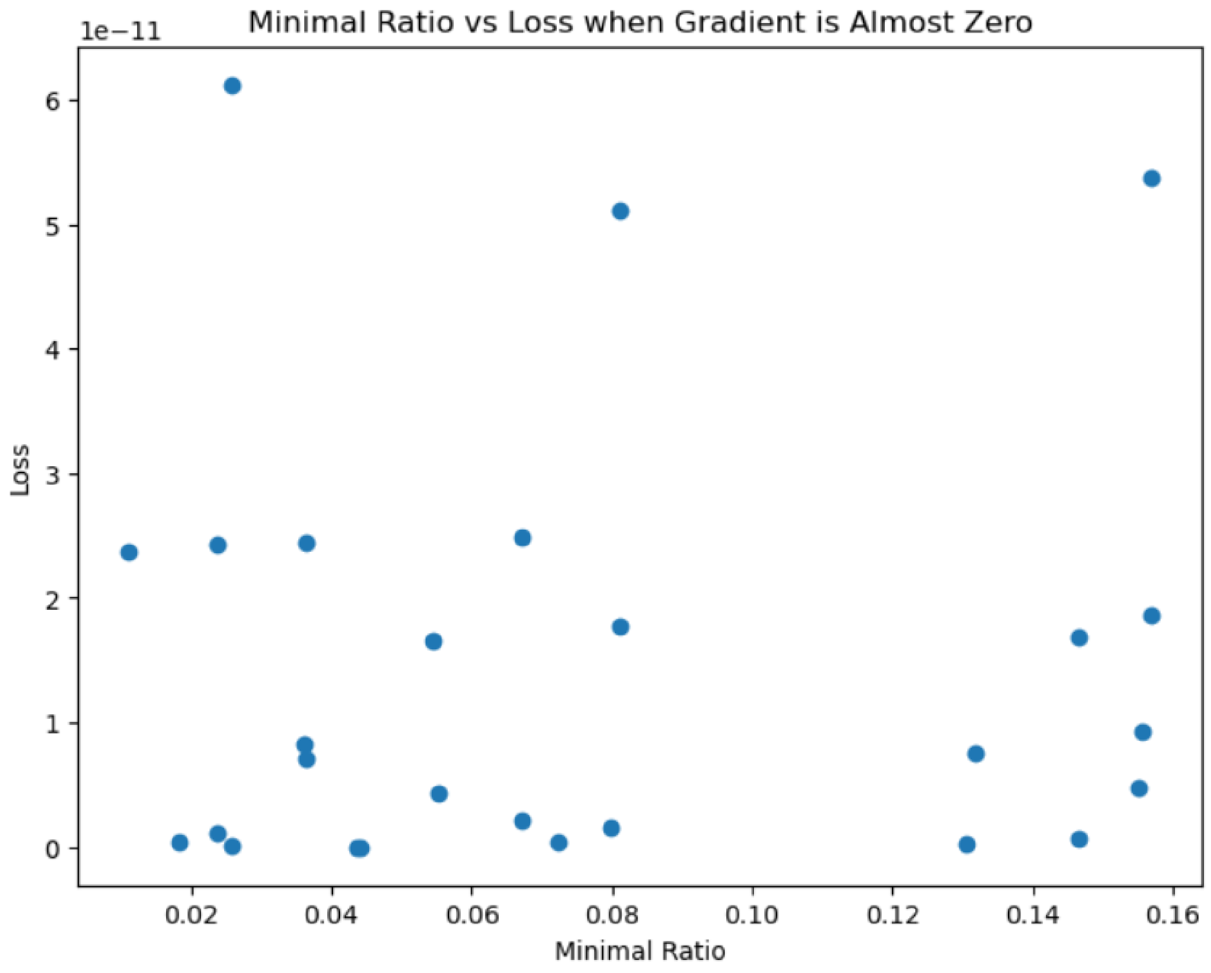
This plot visualizes the relationship between gradient norm and loss during training. The orange line (loss) shows a clear downward trend as training progresses, indicating successful optimization. On the other hand, the blue line (gradient norm) shows significant fluctuations early in training, implying large updates to the weights. As training continues, the fluctuations decrease, but there remains some noise, indicating that the gradient norm does not fully stabilize. This may suggest that while the model is converging in terms of loss, the gradients still vary, possibly due to noisy updates.

2.3. What happens when gradient is almost zero?

Methodology for obtaining weights when gradient norm is zero and the minimal ratio

- A DNN with 3 layers is used. Training data is generated using the function: $\frac{\sin 5\pi x}{5\pi x}$
- The model is trained over multiple epochs, and at each step, the Euclidean or L2 norm of the gradient is calculated.
- A small value closer to zero (5e-5) serves as a switch point from loss minimization to gradient norm minimization.
- During training, when the gradient norm drops below the threshold (near zero), it switches to gradient norm minimization from loss minimization.
- Once the gradient norm is near zero, the current weights are saved by cloning the current parameters of the model.
- The Hessian matrix (approximated by its diagonal in this case) is computed followed by its eigenvalues.
- The minimal ratio is computed by counting the number of positive eigenvalues and dividing this by the total number of eigenvalues.
- In addition, multiple weight configurations are sampled around the critical point (i.e., model parameters when gradient norm is near zero), and the loss is calculated for each sample.
- The proportion of sampled configurations where the loss is greater than the original loss is computed.

Minimal ratio to loss plot



Comments on results

This plot shows the relationship between the minimal ratio and loss when the gradient norm is near zero. The minimal ratio represents the proportion of positive eigenvalues of the Hessian matrix, indicating the degree of local convexity. In this case, the loss tends to be smaller for lower minimal ratios. This suggests that when the local curvature is less positive (flatter region), the model achieves lower losses. However, there is variability, as some points with higher minimal ratios result in relatively low losses. This variability highlights that while a flatter region (lower minimal ratio) can lead to better generalization and lower loss, other factors such as noise or local minima might affect the loss outcome.

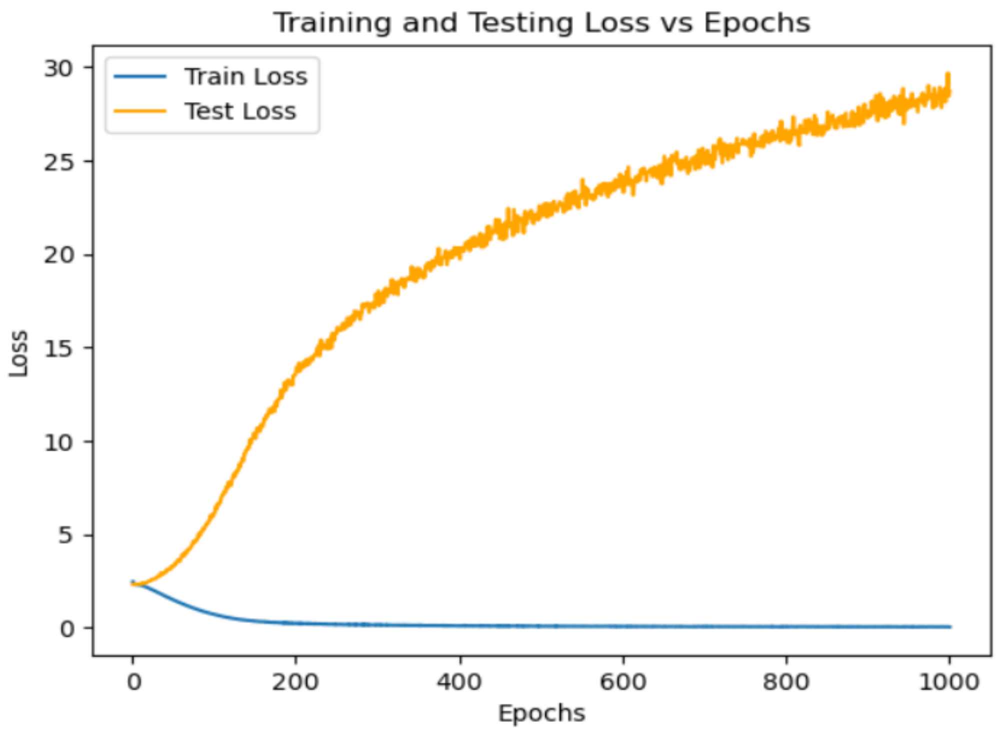
3. **HW1-3**

3.1. Can network fit random labels?

Experiment settings

- Model used: A DNN with 3 hidden layers, each containing 256 nodes
- Loss function used: Cross-entropy
- Task used: MNIST
- Learning rate used: 0.0001
- Number of epochs: 1000
- Batch size: 64
- Optimizer used: Adam

Loss vs epoch plot for both training and testing



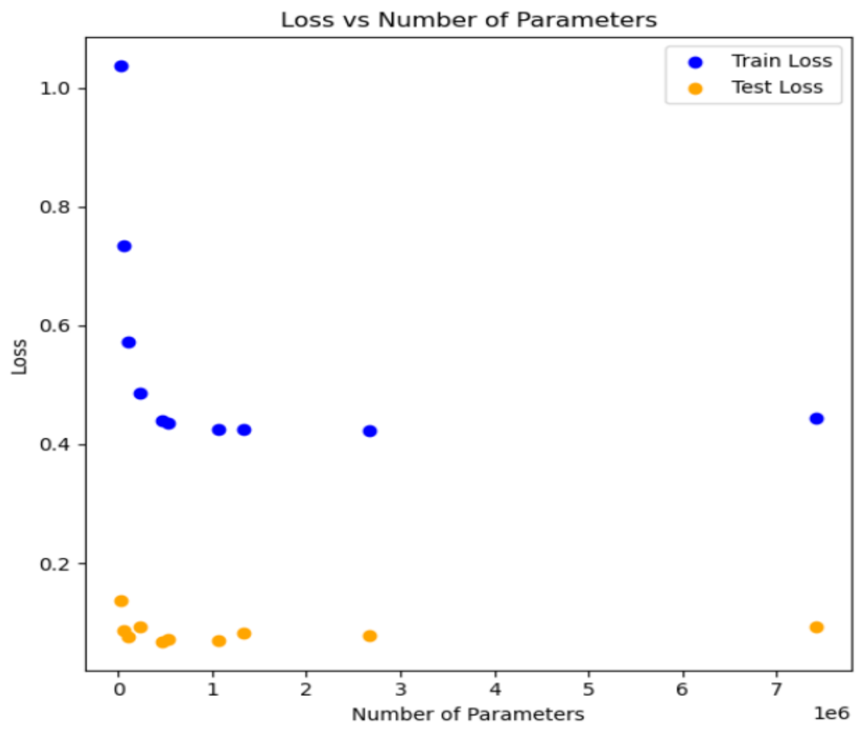
The model is attempting to fit completely random associations between input data and target labels. The decrease in training loss indicates that the model is successfully fitting the random labels, which is expected since neural networks can memorize random patterns if trained long enough. However, the increasing test loss further confirms that the model is failing to generalize, as there is no meaningful relationship to learn due to the randomization of labels.

3.2. Number of parameters vs. generalization

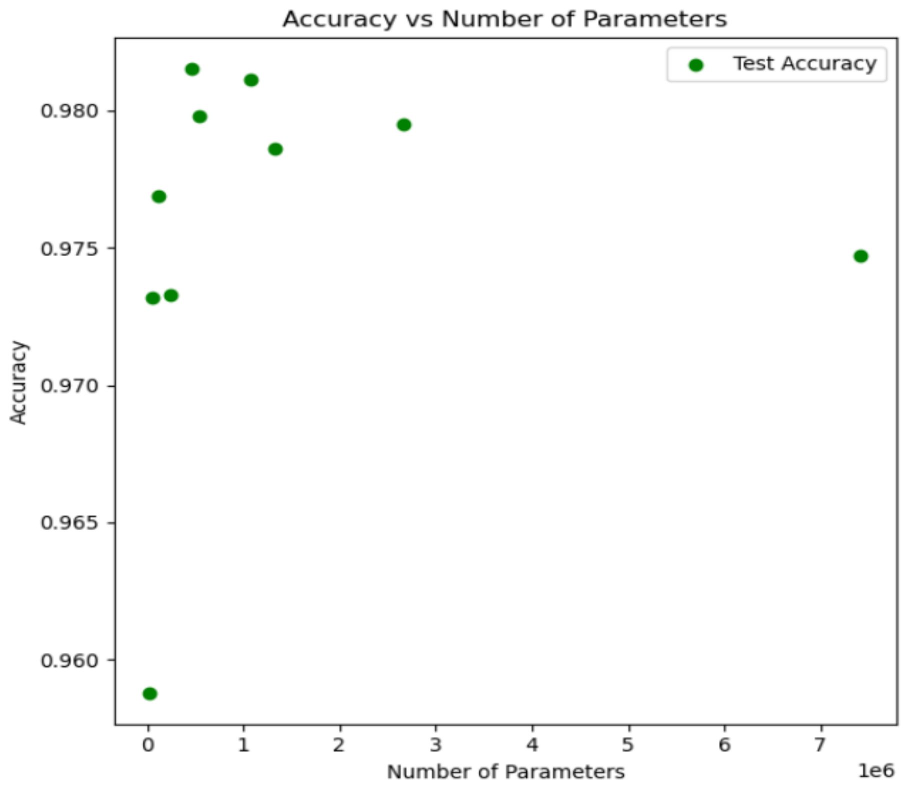
Experiment settings

- Number of models used: 10 DNN models with varying number of parameters:
  - Model 1: Parameters = 52,650
  - Model 2: Parameters = 109,386
  - Model 3: Parameters = 235,146
  - Model 4: Parameters = 535,818
  - Model 5: Parameters = 1,333,770
  - Model 6: Parameters = 25,818
  - Model 7: Parameters = 468,874
  - Model 8: Parameters = 1,068,810
  - Model 9: Parameters = 2,661,898
  - Model 10: Parameters = 7,420,938
- Task: MNIST
- Loss function used: Cross-entropy
- Optimizer used: Adam
- Learning rate used: 0.001
- Number of epochs: 5 epochs for each model
- Batch size: 64 for both training and testing

Loss vs number of parameters plot



Accuracy vs number of parameters plot



Comments on results

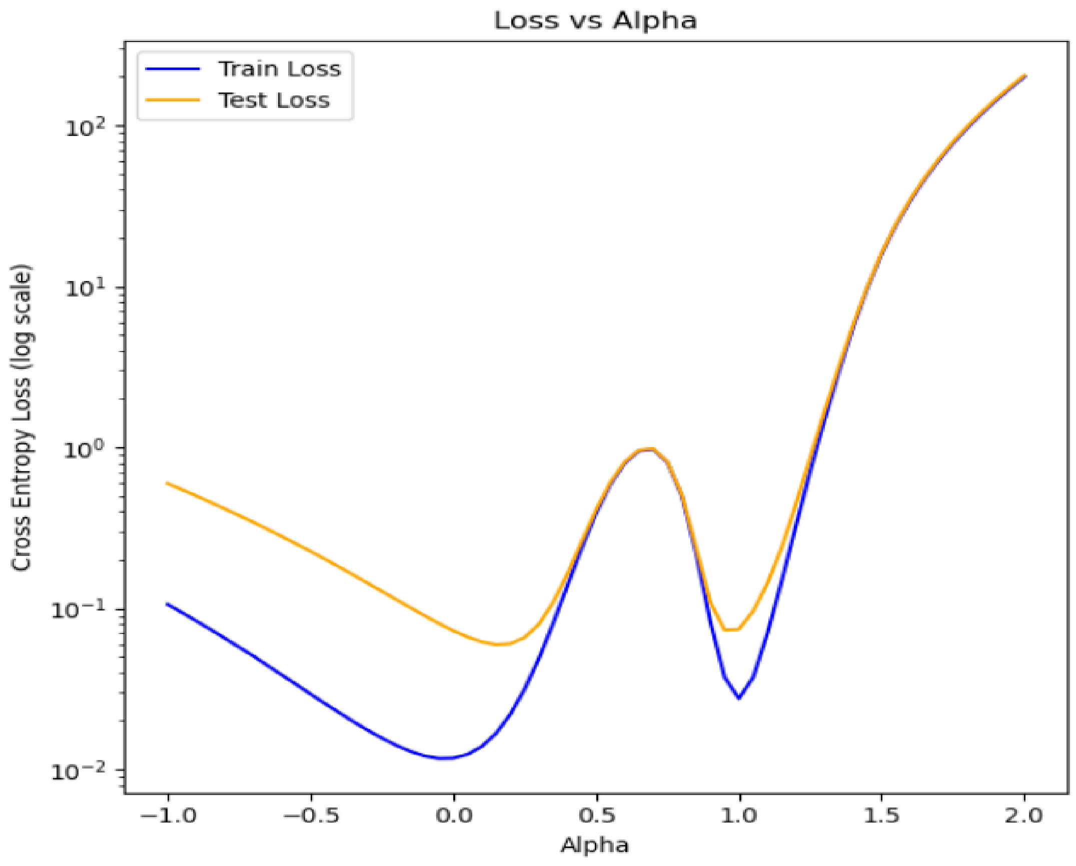
In the results, we can observe that both the training and test losses generally decrease as the number of parameters in the model increases. As for accuracy, there is a clear improvement with the initial increase in model size. However, as the model size exceeds 3 million parameters, test accuracy starts to plateau or even decline slightly, indicating potential overfitting.

3.3. Flatness vs. generalization (part 1)

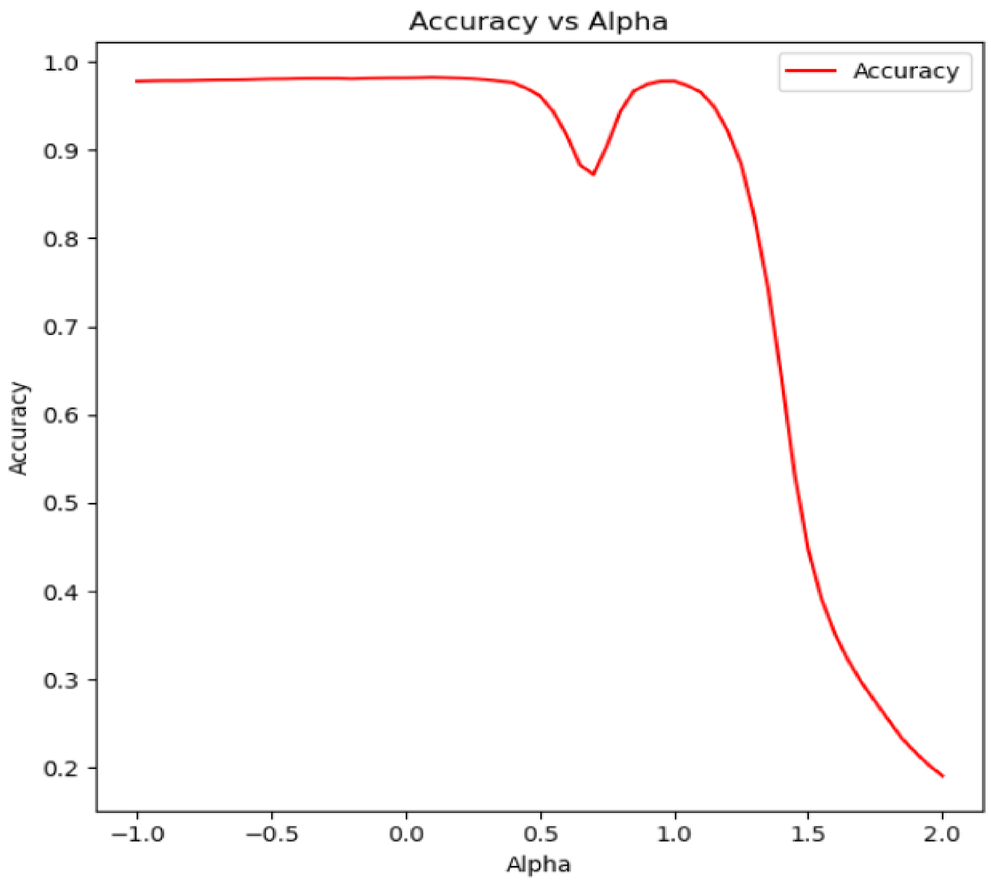
Experimental settings

- Model used: a DNN with 3 layers (256 neurons in each hidden layers).
- Task used: MNIST
- Training approaches:
    - Model 1: Trained with a batch size of 64
    - Model 2: Trained with a batch size of 1024
- Optimizer: Adam optimizer with learning rate of 0.001
- Number of epochs: 10
- Loss function: Cross-entropy
- Interpolation strategy:
    - Model weights are interpolated between Model 1 and Model 2 using different values of alpha (interpolation factor) ranging from -1 to 2
    - The interpolated models are evaluated on both the training and test sets at each interpolation step

Loss vs alpha plot



Accuracy vs alpha plot



Comments on results

In the Loss vs Alpha plot, the training and test losses are shown as a function of the interpolation factor, alpha. The plot demonstrates non-monotonic behavior, with the loss reaching its lowest point around alpha = 0.5 and rising significantly at alpha values near -1 and 2. This suggests that the interpolated models (between two different training setups) exhibit more optimal performance when alpha is near 0.5, which is close to an even blend of the two models.

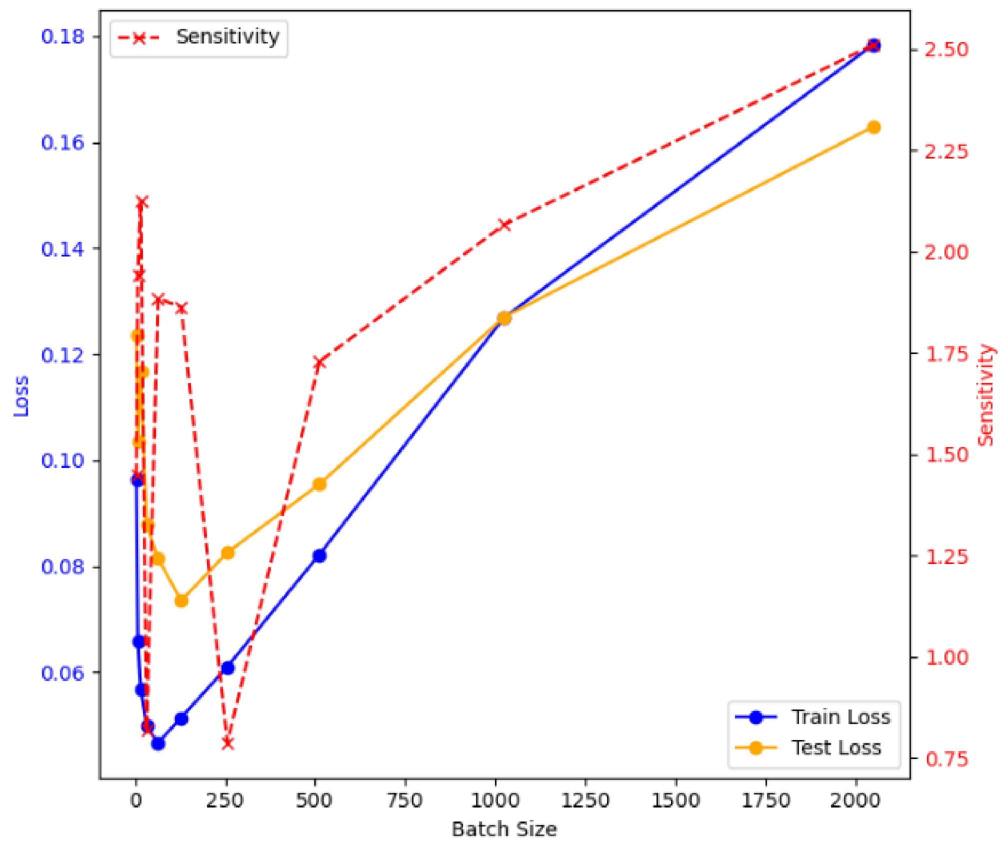
In the Accuracy vs Alpha plot, the model's accuracy remains relatively stable for alpha values between -1 and 1. However, there is a sharp dip around alpha = 0.5, indicating a region where the interpolation results in degraded performance. Beyond alpha = 1, the accuracy drops dramatically, suggesting that excessive blending in the direction of one model leads to significantly poorer results.

3.4. Flatness vs. generalization (part 2)

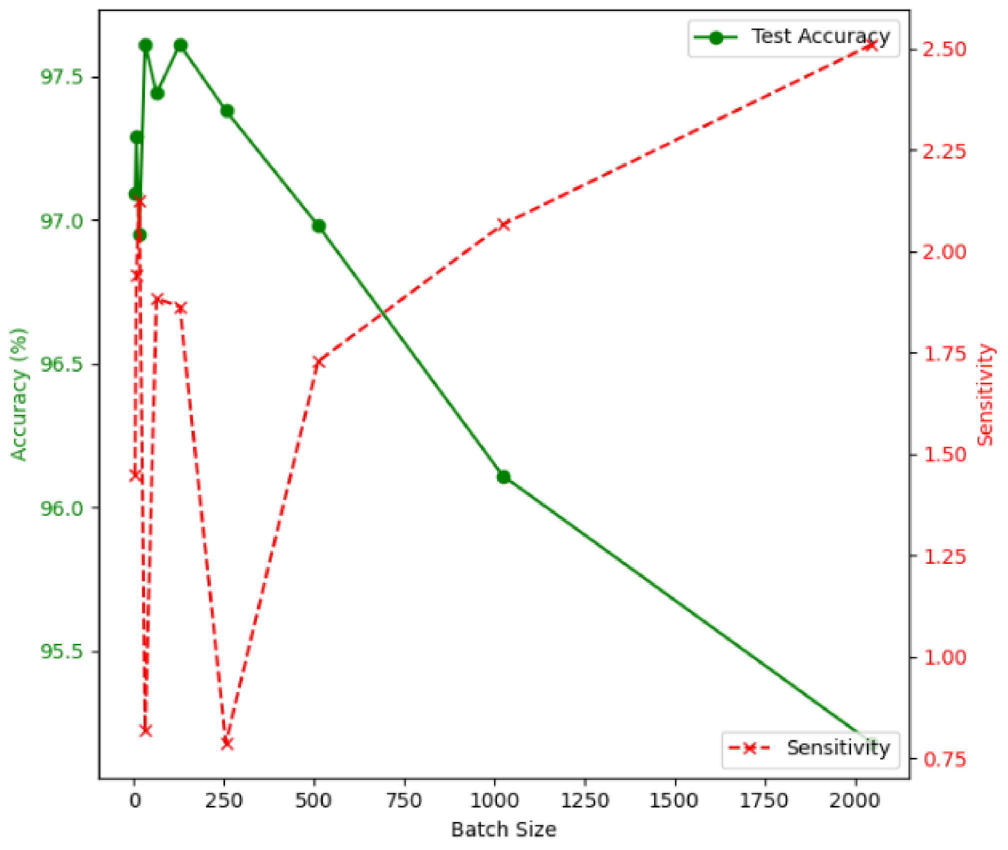
Experimental settings

- Model used: A DNN with 2 hidden layers, each with 128 nodes
- Task used: MNIST
- Loss function: Cross-entropy
- Optimizer: Adam optimizer with a learning rate of 0.001
- Number of epochs: 5 epochs per training session
- Training approaches with varied batch sizes: Varied batch sizes used in the experiments: [4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]
- Sensitivity Calculation: The Frobenius norm of the gradients of the model with respect to the loss function was computed after the forward and backward pass to measure model sensitivity to changes in inputs/parameters

Loss, sensitivity vs. batch size plot



Accuracy, sensitivity vs. batch size plot



Comments on results

The results show that both training and testing loss decrease as the batch size increases up to a certain point, after which they begin to rise again. This suggests that while larger batch sizes initially improve convergence, excessively large batch sizes may hinder the model's ability to fine-tune its learning, leading to higher losses. The accuracy results follow a similar pattern, peaking at moderate batch sizes and then declining as the batch size increases further, indicating that very large batch sizes may lead to underfitting or difficulty in capturing complex data patterns.

In addition, the sensitivity decreases as the batch size increases. Smaller batch sizes tend to have more volatile updates, as indicated by higher sensitivity, whereas larger batch sizes lead to more stable gradient updates but lower sensitivity.