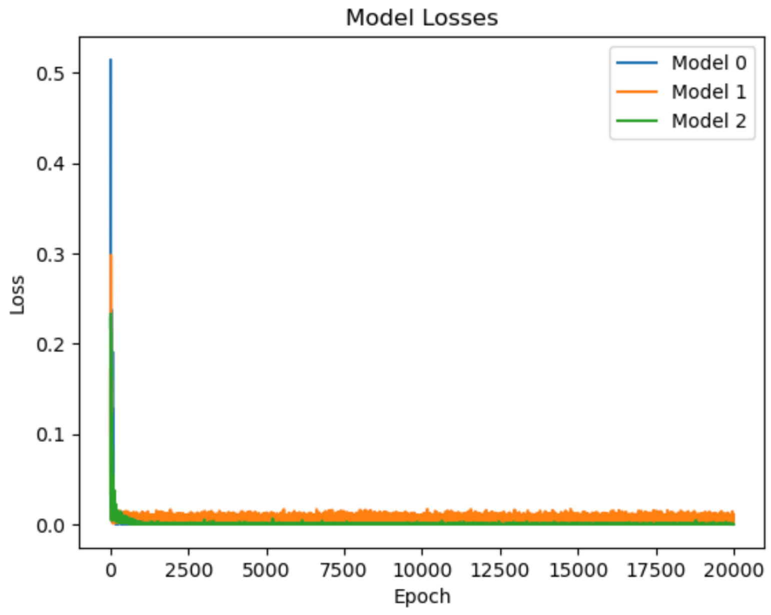
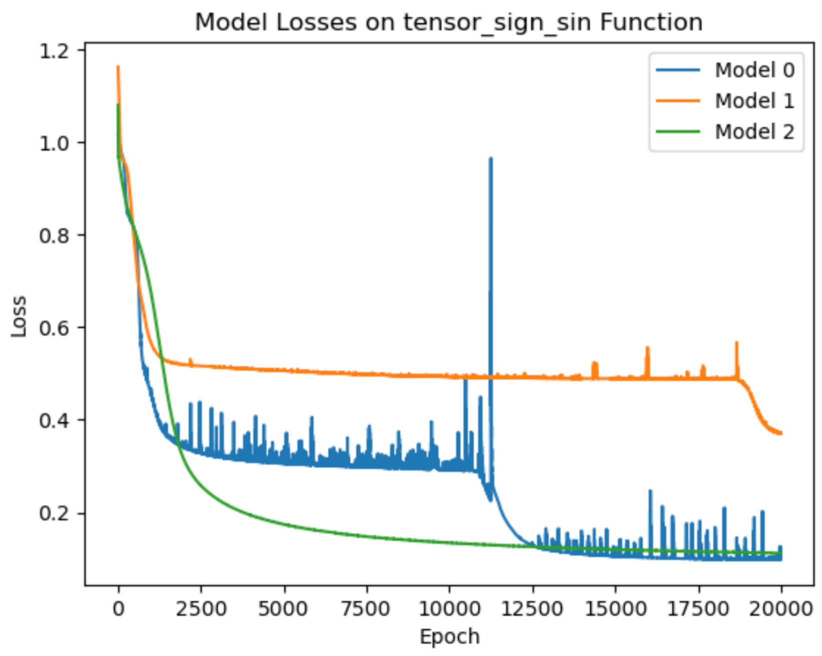
HW1-1 Report Questions

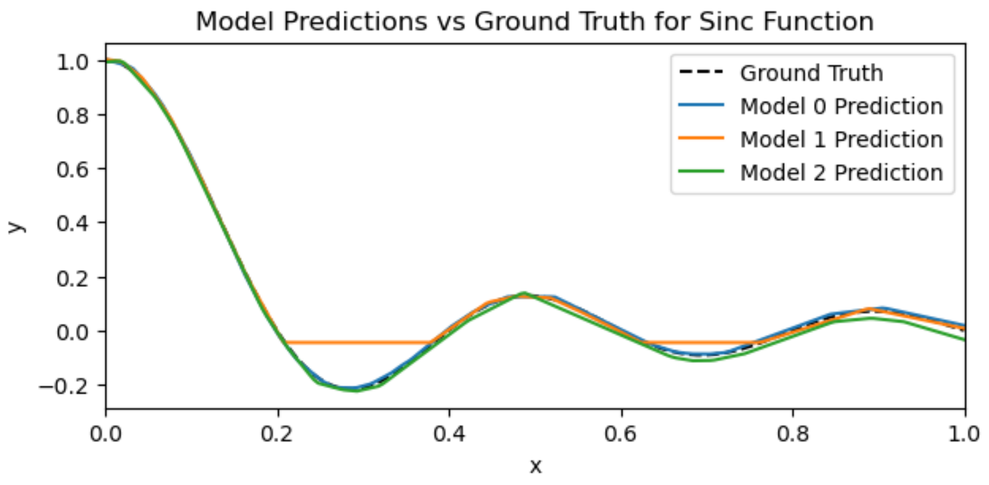
Simulate a Function:

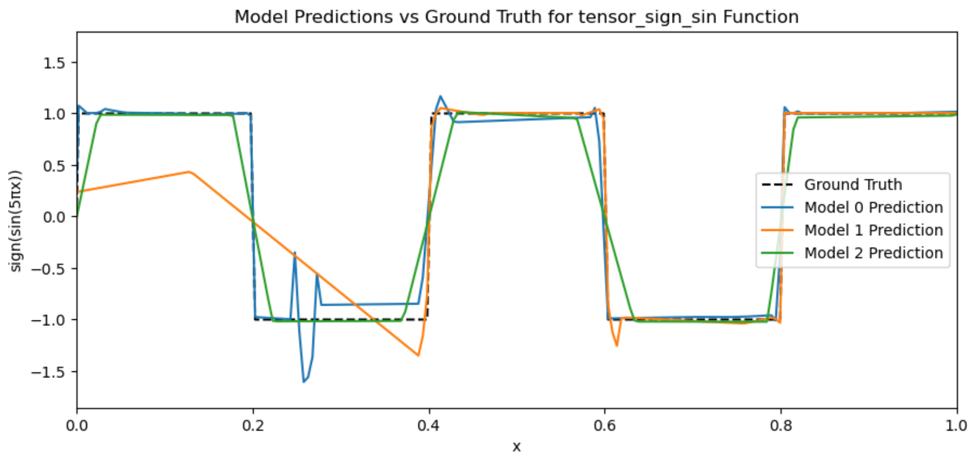
- Describe the models you use, including the number of parameters (at least two models) and the function you use.
    - Model 0: This model is a deep neural network (DNN) with 8 layers. It starts with an input layer that takes 1 feature, followed by 6 hidden layers with varying units (5, 10, 10, 10, 10, 5), and ends with an output layer with 1 unit. Activation functions between layers are ReLU, except for the output layer which has no activation function, suitable for regression tasks. The total number of parameters is 571.
    - Model 1: This model is another DNN with 5 layers. It has an input layer with 1 feature, 3 hidden layers with units (10, 18, 15, 4), and an output layer with 1 unit. ReLU is used as the activation function between layers, with no activation on the output layer. The total number of parameters is 572.
    - Model 2: This is a simpler DNN compared to the first two, with only 2 layers: an input layer with 1 feature that directly connects to a large hidden layer with 190 units, followed by an output layer with 1 unit. ReLU activation is used between the two layers.
    - Simulated two functions:
        - Sinc Function: Defined as **sin(5πx) / (5πx)** with a special case when x=0 to avoid division by zero, handled by **torch.where.**
        - Sign of Sin Function: Defined as sign(sin(5πx)), a piecewise function that returns -1, 0, or 1 based on the sign of sin(5πx).
- In one chart, plot the training loss of all models.





- In one graph, plot the predicted function curve of all models and the ground-truth function curve.





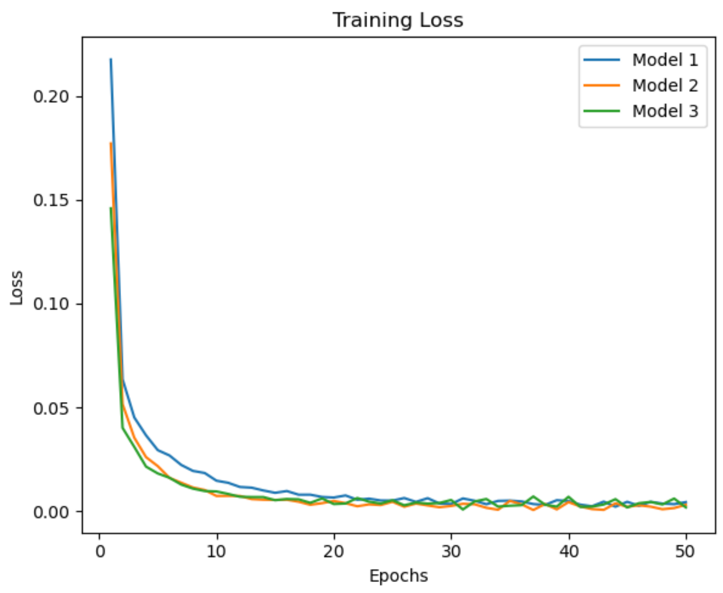
- Comment on your results.
    - For Sicc Function: all three models closely follow the ground truth for the Sinc function, with some minor differences between them. This indicates good model performance on this function.
    - For Sign_Sin function: Model 1 closely matches the ground truth for the tensor_sign_sin function with minor deviations, while Model 0 and Model 2 show significant discrepancies.

Use more than two models in all previous questions. (bonus)

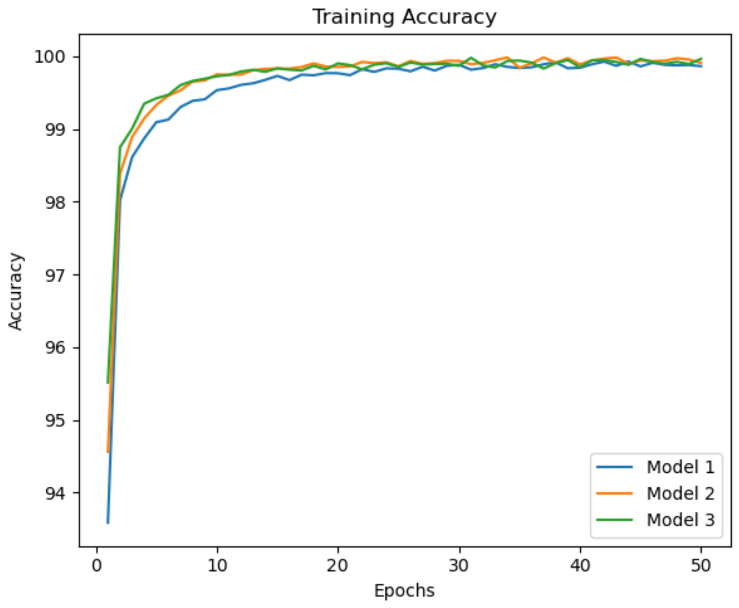
Use more than one function. (bonus)

Train on Actual Tasks:

- Describe the models you use and the task you chose.
    - CNN Model 1: A convolutional neural network (CNN) with two convolutional layers followed by two fully connected layers. The convolutional layers have 10 and 20 output channels, respectively, with a kernel size of 5. The fully connected layers reduce the dimension to 50 and finally to 10, suitable for a 10-class classification task.
    - CNN Model 2: This CNN includes two convolutional layers with 16 and 32 output channels, respectively, and kernel size 3 with padding. It uses two fully connected layers to reduce the dimensionality to 128 and then to 10 for classification.
    - CNN Model 3: An extended version of the previous models with an additional convolutional layer, making three in total with 32, 64, and 128 output channels, respectively. The fully connected layers reduce the dimensionality further to 256 and then to 10 for classification.
- In one chart, plot the training loss of all models.



- In one chart, plot the training accuracy.



- Comment on your results.
    - Training Loss: All models show a decrease in loss over time, indicating learning and improvement in the task. Model 1 starts with the highest loss but its rate of improvement is rapid, suggesting good learning efficiency. Model 2 and Model 3 start with lower initial losses and converge to a similar value, with Model 3 slightly lower, indicating possibly better performance.
    - Training Accuracy: The accuracy for all models increases over epochs, which is a good sign of model performance. Model 1 shows a steep improvement in accuracy, aligning with the observed loss decrease. Model 2 and Model 3 have a similar trajectory with Model 3 reaching a marginally higher accuracy, suggesting it might be the best-performing model among the three.

Use more than two models in all previous questions. (bonus)
Train on more than one task. (bonus)

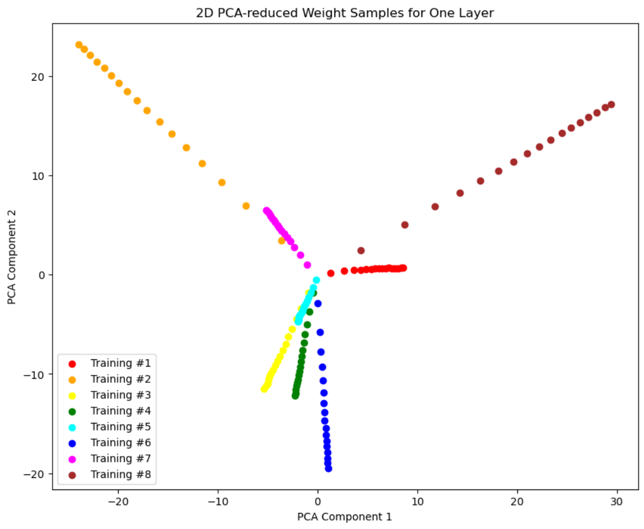
HW1-2 Report Questions

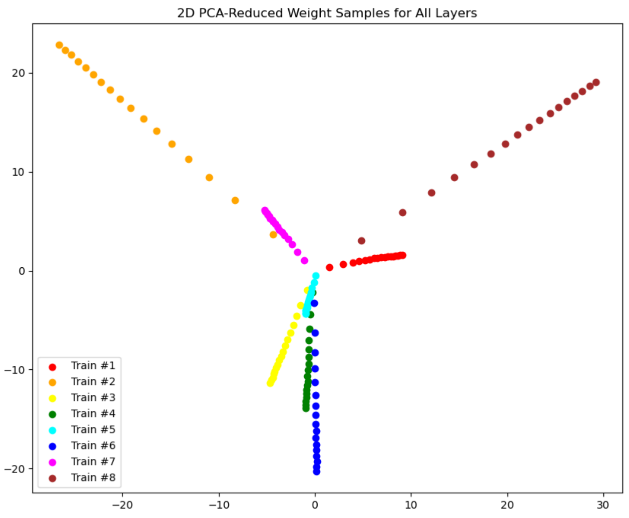
Visualize the optimization process.

- Describe your experiment settings. (The cycle you record the model parameters, optimizer, dimension reduction method, etc)
    - The experiment settings employs a structured approach to training a Deep Neural Network on the MNIST dataset, capturing model parameters and gradients every 3 epochs to analyze learning dynamics using PCA for dimensionality reduction. With the Adam optimizer for efficient convergence and a simple yet effective three-layer architecture, the setup aims to visualize weight trajectories in a 2D space, enhancing understanding of the model's evolution. The training involves multiple runs to observe learning variance, employing data normalization and

batch processing for optimized learning, although it lacks certain best practices like validation and early stopping.

- Train the model for 8 times, selecting the parameters of any one layer and whole model and plot them on the figures separately.

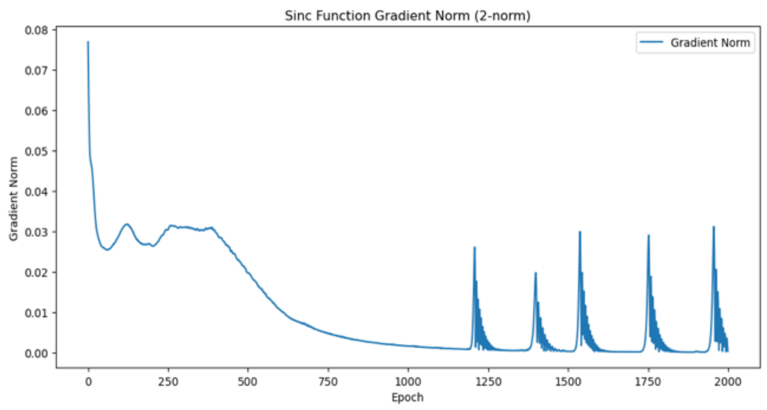


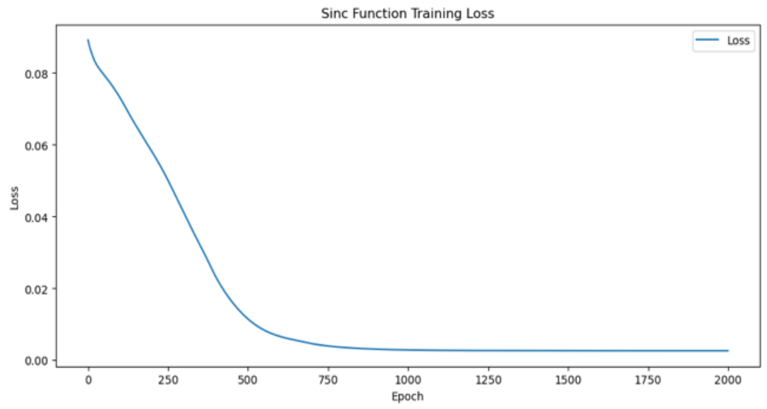


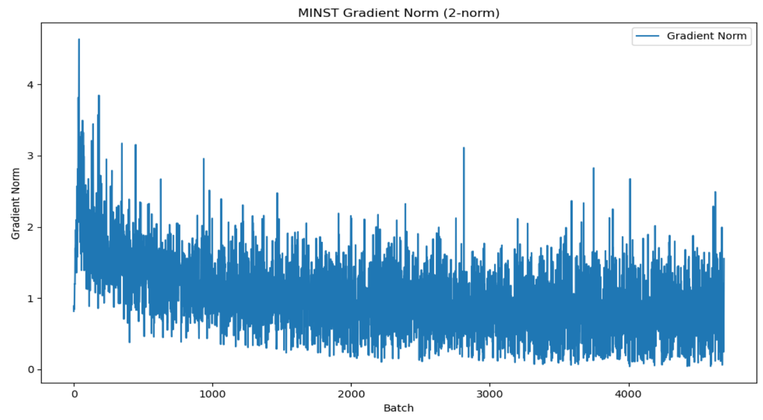
- Comment on your result.
    - Learning Trajectory: The plots show a clear progression of weight adjustments from early to later training epochs, suggesting the network's learning process is active and evolving. Convergence Indication: The points cluster more closely in later epochs, which typically indicates the network's weights are starting to converge. Layer Difference: The first plot for one layer shows a more erratic progression, while the second plot for all layers indicates a smoother overall learning trajectory. Principal Component Variance: The spread along the first principal component is greater than the second, implying it captures more variance in the weights.

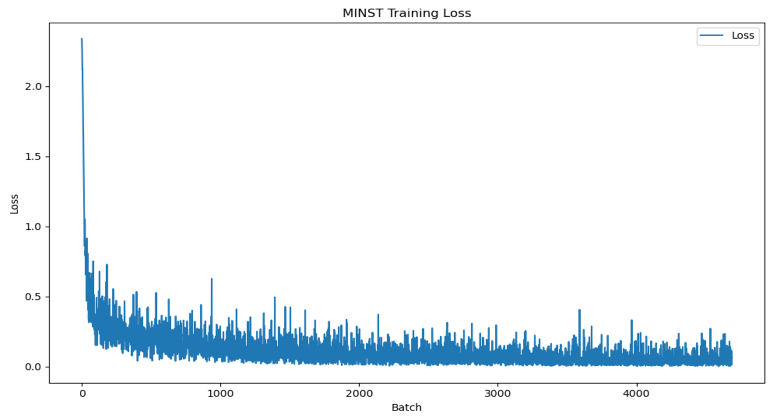
Observe gradient norm during training.

- Plot one figure which contain gradient norm to iterations and the loss to iterations.


Sinc Function Gradient Norm (2-norm)
Gradient Norm
Gradient Norm
Epoch


Sinc Function Training Loss
Loss
Loss
Epoch

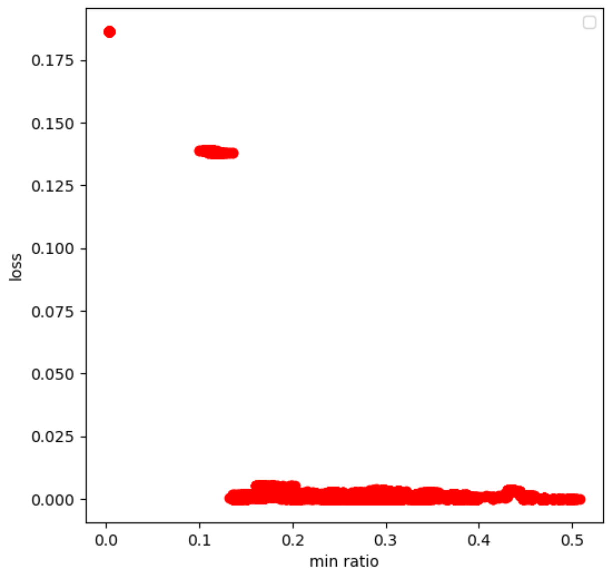

MINST Gradient Norm (2-norm)
Gradient Norm
Gradient Norm
Batch


MINST Training Loss
Loss
Loss
Batch

- Comment your result.
    - The training plots indicate that the model is generally learning well. The gradient norm plot shows initial rapid learning followed by stability with occasional spikes, which might need further investigation. The loss plot shows a smooth and consistent decrease in training loss, suggesting good convergence without overfitting. However, validation on unseen data is essential to ensure the model's generalization.

What happens when gradient is almost zero?

- State how you get the weight which gradient norm is zero and how you define the minimal ratio.
    - In this process, a two-stage training process is applied to a deep neural network using PyTorch, where the first stage involves conventional training with backpropagation to update model parameters using an Adam optimizer. The second stage involves a more nuanced approach where, after each optimization step, the script checks for parameters with gradients close to zero by calculating the L2 norm of the concatenated gradients of all parameters. When these gradients are near-zero, indicating a local minimum or plateau, the script computes the Hessian matrix to gain insight into the curvature of the loss landscape at that point. The "minimal ratio" is then defined as the ratio of positive eigenvalues of the Hessian to the total number of eigenvalues, providing a measure of the convexity around the current parameters, with a higher ratio indicating a more stable and convex minimum. This approach aims to identify points in the parameter space where the model is at a stable minimum by considering both first-order (gradient) and second-order (curvature) information. The final part of the script involves plotting these minimal ratios against the loss values to analyze the relationship between the stability of minima and the performance of the model, potentially highlighting the trade-offs between reaching lower loss values and finding stable minima in the parameter space.
- Train the model for 100 times. Plot the figure of minimal ratio to the loss.



- Comment your result.

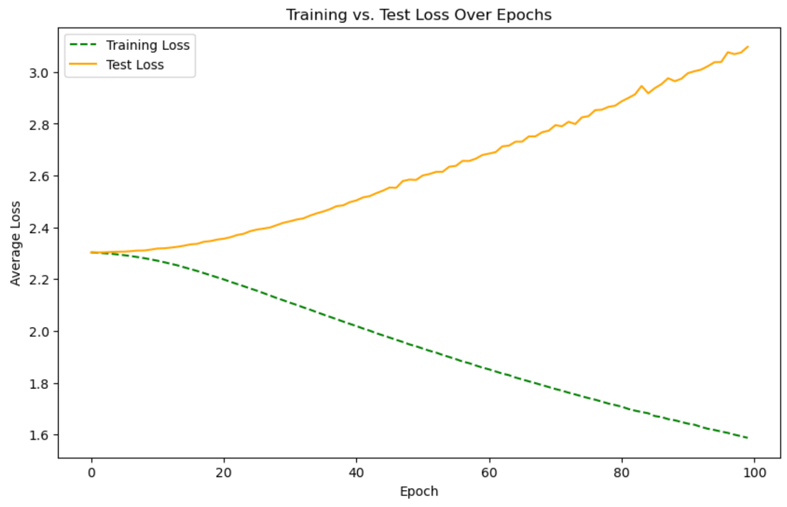
Bonus

○Use any method to visualize the error surface.

○Concretely describe your method and comment your result.

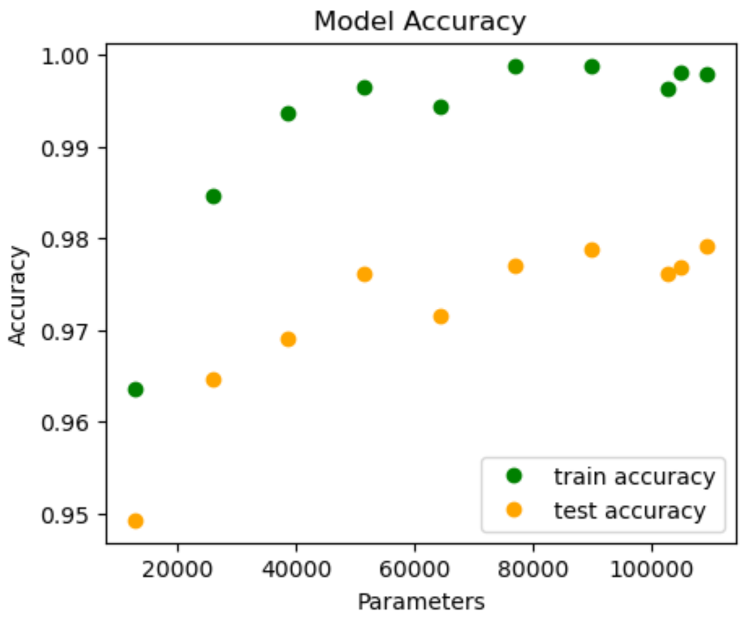
HW1-3 Report Questions

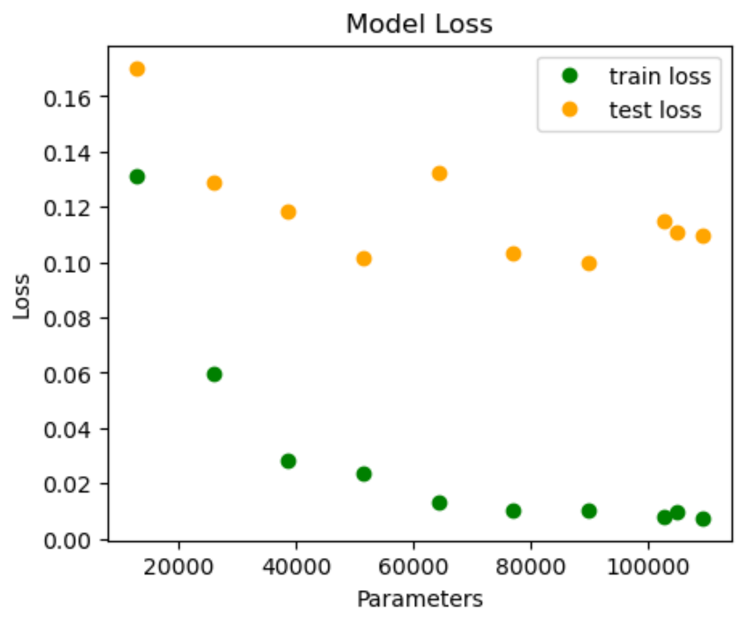
Can network fit random labels?

- Describe your settings of the experiments. (e.g. which task, learning rate, optimizer)
    - This experiment involves training a neural network for a classification task using PyTorch. The model consists of a flattening layer, two dense layers with ReLU activation, and an output layer. It uses the Adam optimizer with a learning rate of 1×10^{-4}, Cross-Entropy Loss, and is trained for 100 epochs on a GPU. Training progress is monitored using a progress bar, and both training and test losses are tracked and reported after each epoch.
- Plot the figure of the relationship between training and testing, loss and epochs.
    - 

Number of parameters v.s. Generalization

- Describe your settings of the experiments. (e.g. which task, the 10 or more structures you choose)
    - This part sets up an experiment to evaluate the performance of ten different neural network models on an image classification task, likely involving 28x28 pixel images such as those found in the MNIST dataset. The models vary in complexity, with the primary differences being the number of neurons in their layers: Model0: A basic model with a single hidden layer of 16 neurons. Model1 to Model7: Progressively larger models with increasing neuron counts in the first hidden layer, ranging from 32 to 128 neurons, and a consistent second hidden layer of 16 neurons. Model8 and Model9: More complex models with the first hidden layer of 128 neurons followed by larger second hidden layers of 32 and 64 neurons, respectively.
    - Each model is trained for 20 epochs using the Adam optimizer and cross-entropy loss. The experiment measures the number of parameters, training and test losses, and accuracies on both datasets to assess how the complexity of each model affects its learning and generalization capabilities.
- Plot the figures of both training and testing, loss and accuracy to the number of parameters.





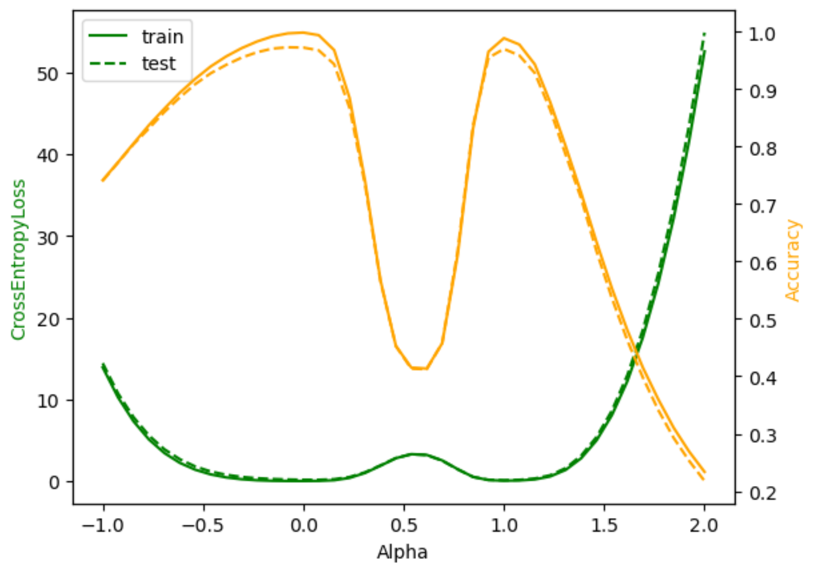
- Comment your result.
    - It appears that there are two plots, one showing model accuracy and the other showing model loss, both in relation to the number of parameters in the models. For model accuracy, the first plot displays a generally increasing trend in training accuracy as the number of parameters increases, which indicates that more complex models with a larger number of parameters tend to fit the training data better. However, the test accuracy does not follow the same trend; it increases initially with the number of parameters but then plateaus or slightly decreases. This could suggest that as models become too complex, they might be overfitting to the training data, which means they perform well on the training set but not as well on unseen data (test set).
      For model loss, the second plot shows the loss of the models during training and testing. The training loss consistently decreases as the number of parameters increases, which again suggests that models with more capacity fit the training data more closely. In contrast, the test loss decreases initially but then starts to increase after a certain point. This is a classic indication of overfitting, where the loss for unseen data begins to rise as the model starts to memorize the training data rather than learning generalizable patterns.
    - The plots suggest that there is a sweet spot in model complexity that balances the trade-off between underfitting and overfitting. Models that are too simple do not capture all the patterns in the data, leading to higher bias, while models that are too complex capture too much noise from the training data, leading to higher variance. The optimal model is likely to be one that has enough parameters to learn the underlying structure of the training data without capturing the noise, which would be indicated by a high test accuracy and a low test loss. It might be

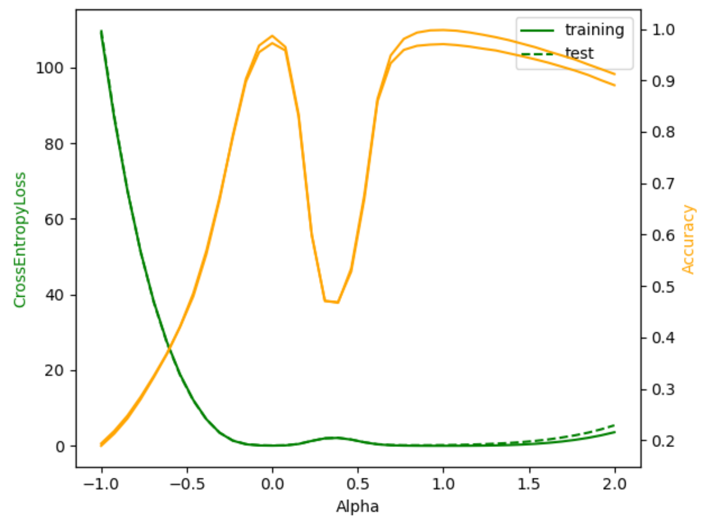
beneficial to implement techniques like regularization, cross-validation, or model ensembles to improve generalization for the more complex models.

Flatness v.s. Generalization

○Part 1:

- Describe the settings of the experiments (e.g. which task, what training approaches)
    - This part code conducts a series of experiments on the MNIST dataset using a simple neural network with the objective of examining the effects of different batch sizes and learning rates on model training. It compares the performance of models trained with batch sizes of 64 and 1024, and learning rates of 1e-3 and 1e-2, over 40 epochs. Additionally, it explores the loss landscape by interpolating between models with varying hyperparameters, and evaluates the impact on test and training loss and accuracy. This experimentation aims to reveal insights into how such hyperparameters influence the training process and the optimization characteristics of the network.
- Plot the figures of both training and testing, loss and accuracy to the number of interpolation ratio.



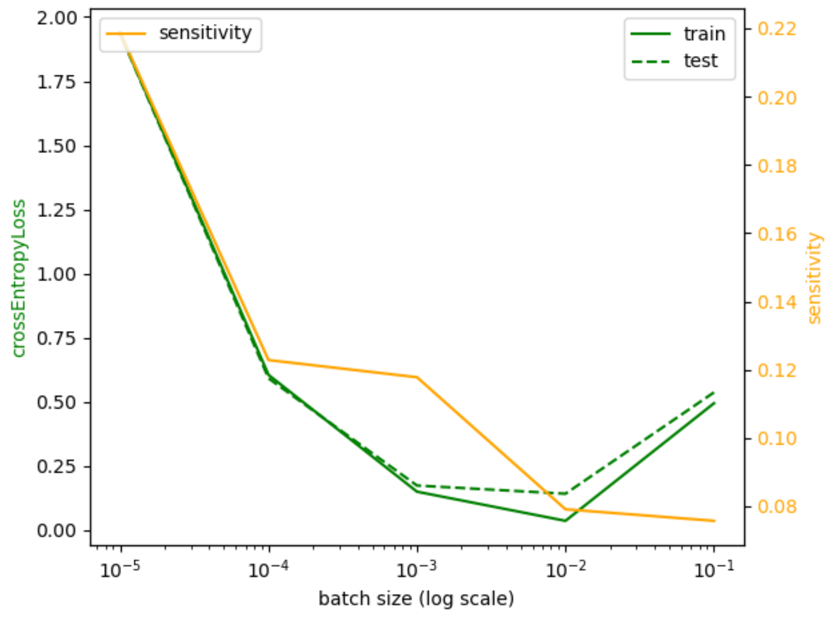


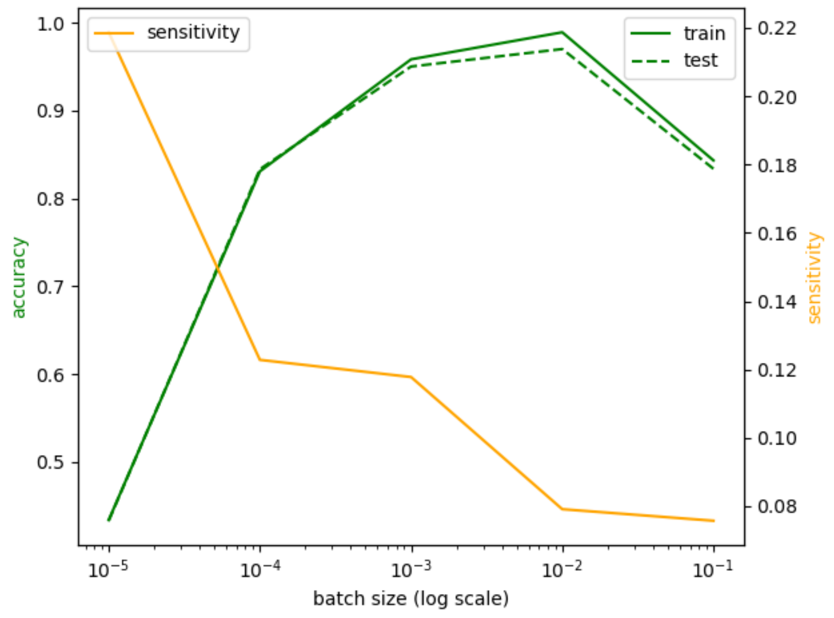
- Comment your result.
  - The graphs presented illustrate the effects of model interpolation on the MNIST dataset, where two distinct trends emerge across the range of alpha values. For values of alpha less than 0.5, both training and test losses are relatively low, with training accuracy high, suggesting that models in this range generalize well to the training data and to a lesser extent, also to the test data. However, as alpha exceeds 0.5, there is a dramatic increase in loss and a corresponding decrease in accuracy, particularly pronounced on the test set, indicating a significant overfitting issue where the model fits the training data well but fails to generalize to unseen data. This sharp drop in performance suggests that the interpolated models with higher alpha values likely correspond to a less favorable part of the parameter space, leading to solutions that are highly specific to the training set and do not translate well to the test set, underlining the critical importance of choosing models that balance performance on both training and unseen data for robust prediction capabilities.

○Part 2 :

- Describe the settings of the experiments (e.g. which task, what training approaches)
  - The experiment involves training a neural network model on the MNIST dataset to classify handwritten digits using different learning rates, ranging from 1e-1 to 1e-5. It employs a large batch size of 1024 and a three-layer architecture with ReLU activations. Over 40 epochs, the model's performance is evaluated in terms of training and test loss, accuracy, and sensitivity to input changes, which measures the model's robustness. The aim is to understand the impact of learning rate variations on the model's ability to generalize and its response to data variations, ultimately to identify the optimal learning rate for such image classification tasks. The training leverages GPU acceleration, indicated by the \`.to(device)\` calls, ensuring efficient processing and evaluation.

- Plot the figures of both training and testing, loss and accuracy, sensitivity to your chosen variable.





- Comment your result.
  - The graphs demonstrate the effects of batch size on a neural network's performance, with batch size presented on a logarithmic scale against cross-entropy loss and accuracy for both training and test datasets. As batch size increases, both training and test losses decrease initially, indicating improved learning; however, beyond a certain batch size, the test loss starts to rise, signaling overfitting. Similarly, the training accuracy remains high across batch sizes, but test accuracy shows a modest decline after surpassing an optimal batch size, suggesting that the model begins to overfit to the training data and loses

generalization capability. The sensitivity, a measure of the model's responsiveness to input changes, decreases steadily with larger batch sizes, reflecting a trade-off where larger batches result in more stable but potentially less flexible models. These trends suggest there is an optimal batch size that strikes a balance between learning efficiency, generalization to new data, and model sensitivity, which is crucial for model tuning to prevent overfitting and ensure robust performance on unseen data.

○Bonus : Use other metrics or methods to evaluate a model's ability to generalize and concretely describe it and comment your results.