Course: 8430 (Deep Learning)

Name: Vinay Ponugoti
CUID: C32054414
Email: vponugo@g.clemson.edu
GitHub Link: https://github.com/Vinay-ponugoti/DeepLearning.git

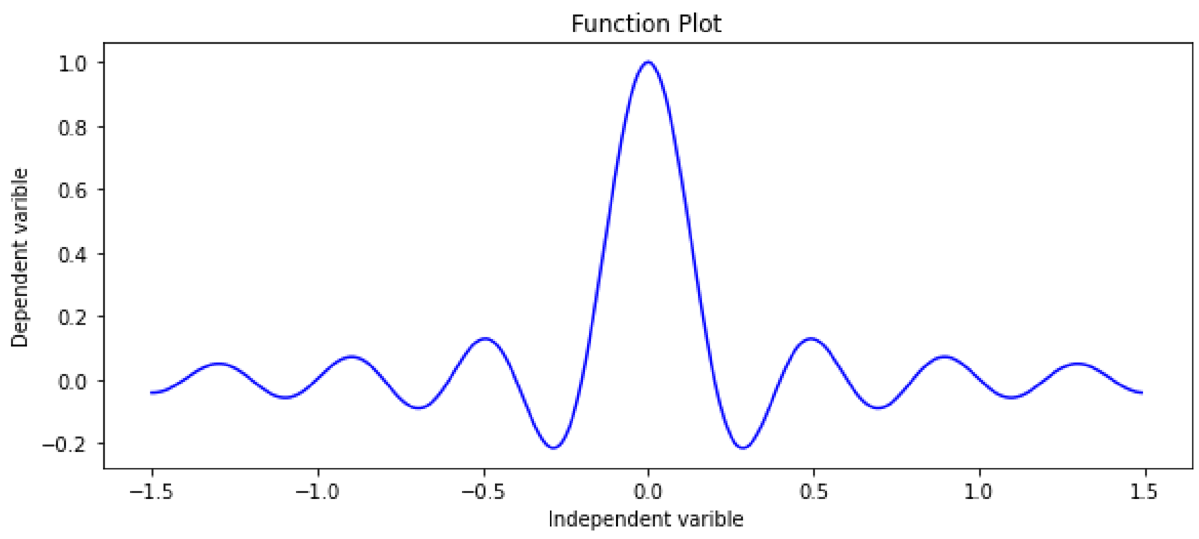
HW1-1: Deep vs Shallow

Three different models, each with its own architecture, were trained on two distinct functions. The models referred to as model1, model2, and model3, are fully connected neural networks consisting of seven, four, and one layer, respectively. They have 571, 572, and 572 parameters. Interestingly, the average number of nodes differs across the layers in each model. For all three models, the learning rate was established at 0.001.

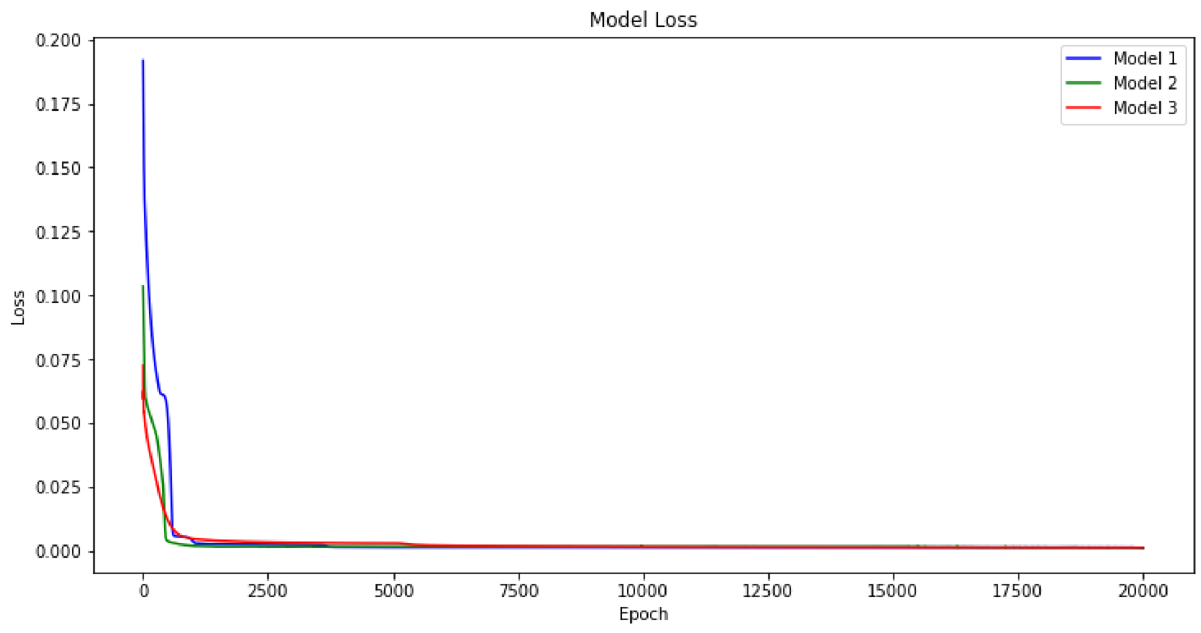
Simulated Functions:
1) y=(sin(5*(pi*x)))/((5*(pi*x))
2) y=sgn(sin(5*pi*X1))

Function 1[**y=(sin(5*(pi*x)))/((5*(pi*x))**]

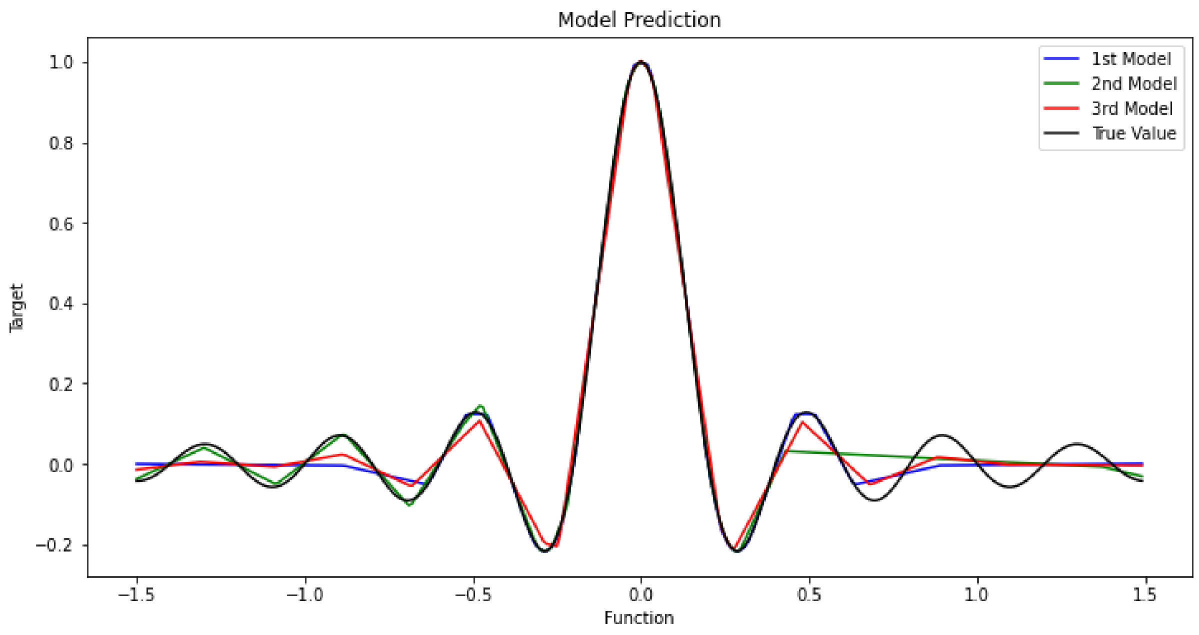
Graph:



The graph below shows the relationship between loss and epoch after training the three models on function 1.



The graph below shows the prediction for the three models:

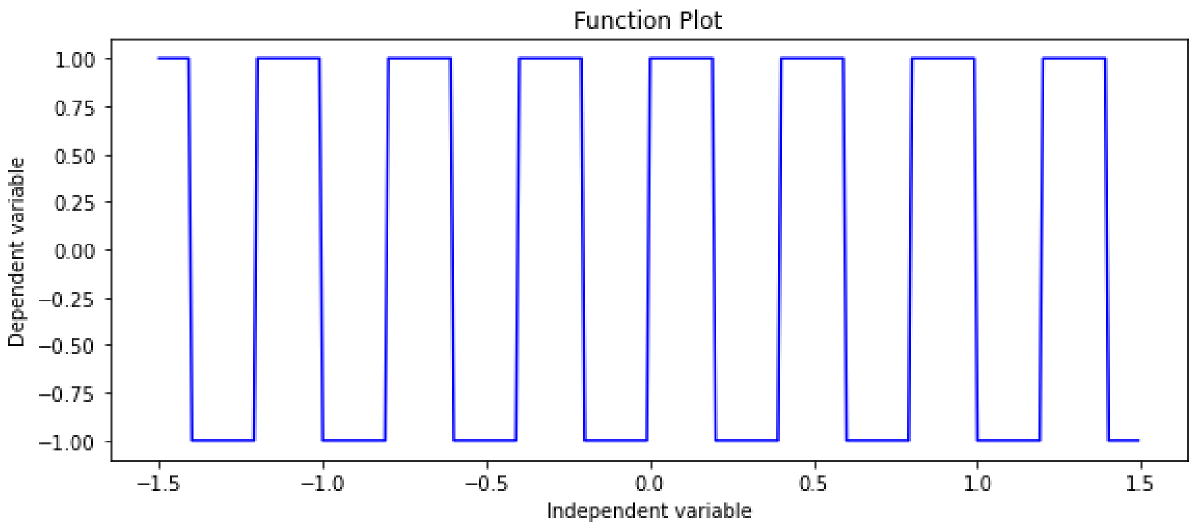


The "NN Deep" model converged more quickly than the other models. After the first 1000 epochs, the "NN Shallow" model consistently showed a higher loss in every epoch. This indicates that deeper models are able to learn more rapidly and effectively than their shallower counterparts.

Course: 8430 (Deep Learning)

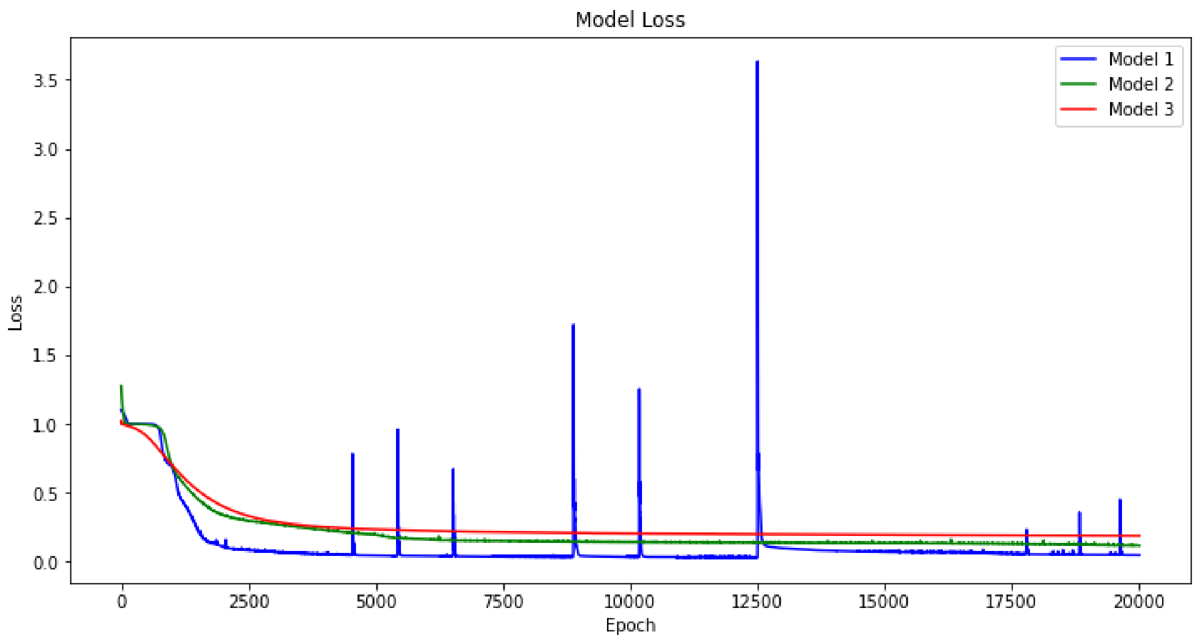
Function 2:- **y=sgn(sin(5*pi*X1))**

X and Y are shown as functions in the figure below.



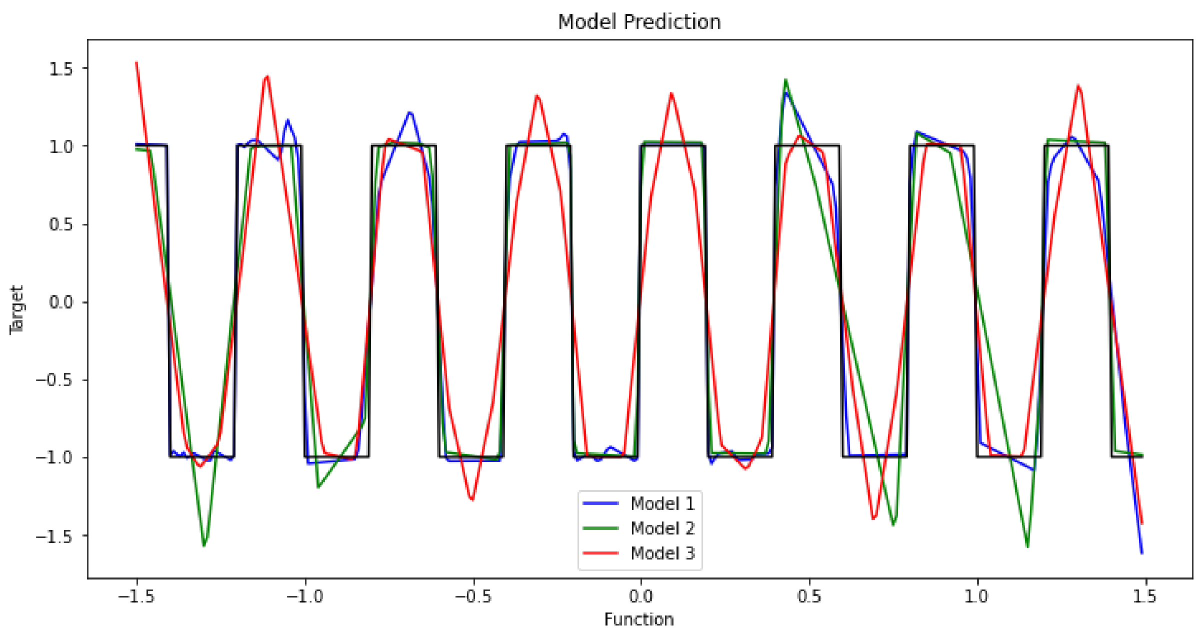
In simulating the function sgn(sin(5*pi*x) / (5*pi*x), it's typical for all models to achieve stability after a set number of generations.

The function chart below illustrates the loss associated with each of these models.



The final figure below shows the actual and predicted values of the function's sign, represented by the expression sin(5*pi*X1).

Course: 8430 (Deep Learning)



Result: All three models found the function challenging, and they all hit the maximum number of epochs before converging. Each model1, model2, and model3 realized that the accuracy function could produce the desired results when dealing with an unknown input. The deeper models showed fewer errors at the edges of the plot, while those situated in the middle performed well overall.

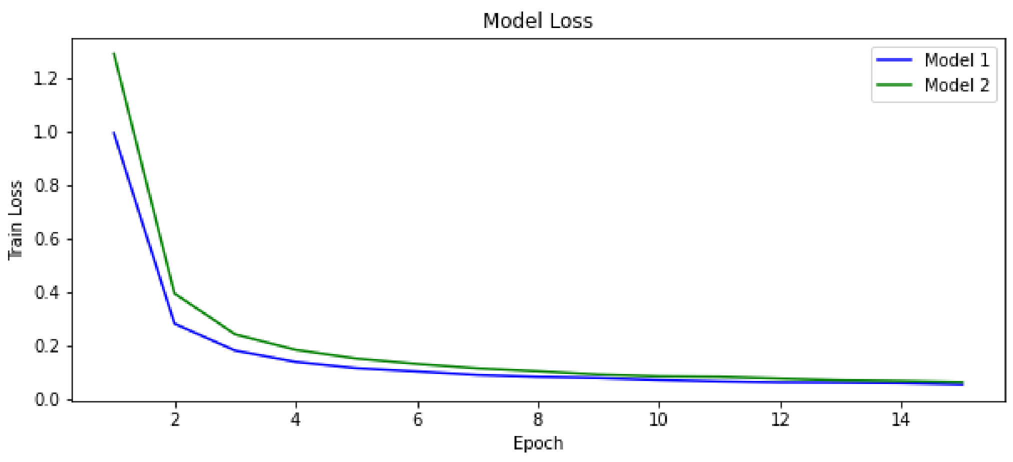
HW 1-1 Train on Actual Tasks

The models presented below were trained using the MNIST dataset. The training set consists of sixty thousand images, while the test set contains ten thousand images. I used convolutional neural networks (CNN) with a batch size of ten to make two models.

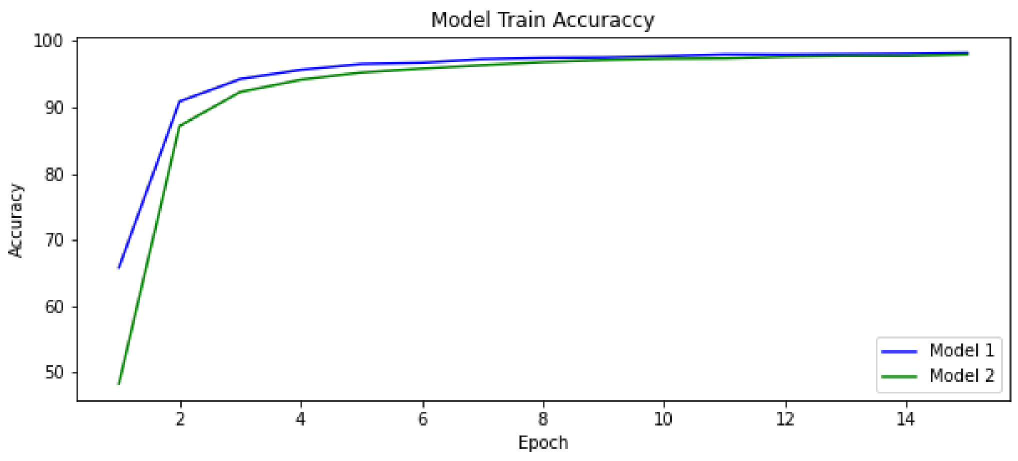
Both models are fully interconnected, and to share data with the activation function, I employed the max pooling function. To achieve this, I used a 2D convolutional layer along with 2D max pooling and set the learning rate to infinity. Each layer of the network has a different number of nodes.

A comparison of the training loss for Model 1 and Model 2 is shown in the figure below.

Course: 8430 (Deep Learning)



The diagram below illustrates the accuracy of two models during the training process, showcasing their performance on both training and test sets.



Result: Model 1 is an optimized CNN model that outperforms Model 2 in terms of efficiency and loss. When we compare the two models against the test data, we observe that each model tends to perform better than the other on the training data. This behavior aligns with our expectations.

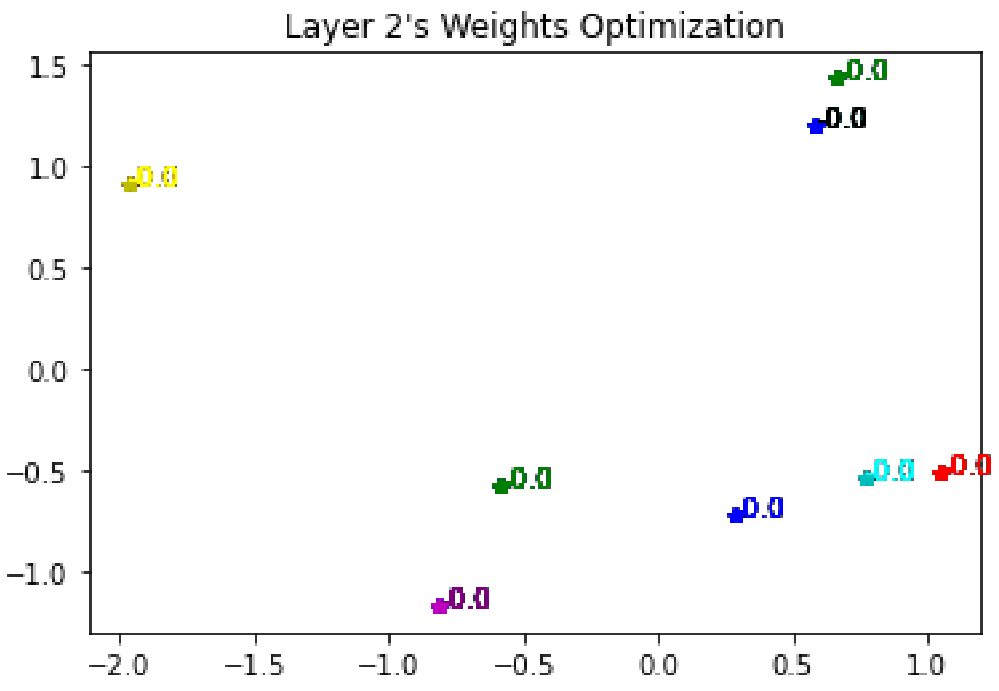
Course: 8430 (Deep Learning)

HW1-2: Optimization

Task1: Visualize the optimization process

To train the function sin(5*(pi*x))/((5*(pi*x))), It was a deep neural network with three fully connected layers that I used. To effectively excite the function, I utilized 57 different parameters.

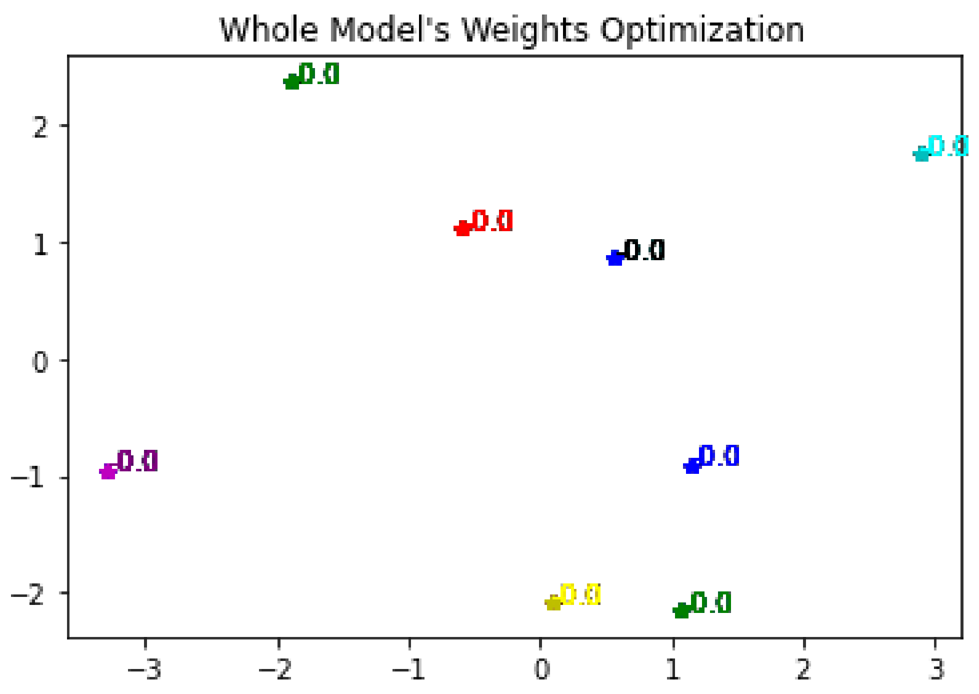
Network Layers and displays the 2nd layer of the network.



The Adam optimizer was used to enhance the network's performance. Over the course of eight separate training sessions, each lasting 30 epochs, the model's weights were regularly documented. PCA was applied to effectively reduce the dimensionality. A consistent learning rate of 0.001 was maintained across all models.

Both of these illustrations demonstrate how weight optimization happens through backpropagation as the network learns from new information during training. The figures emphasize the gradual fine-tuning and optimization of the weights.

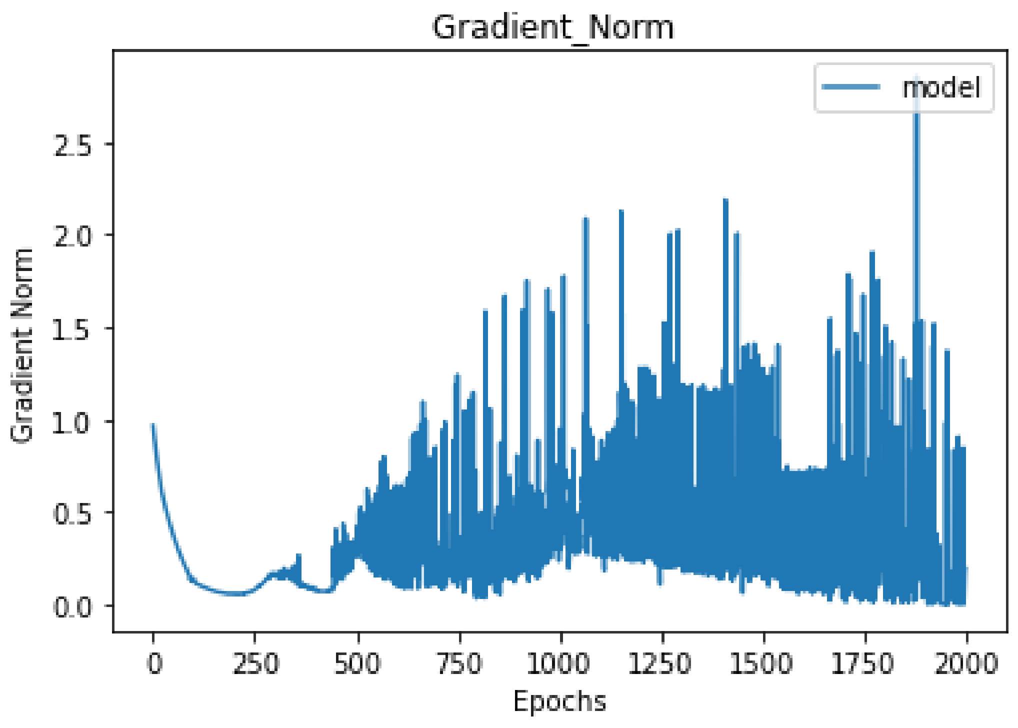
Course: 8430 (Deep Learning)



The above image shows the whole model optimization

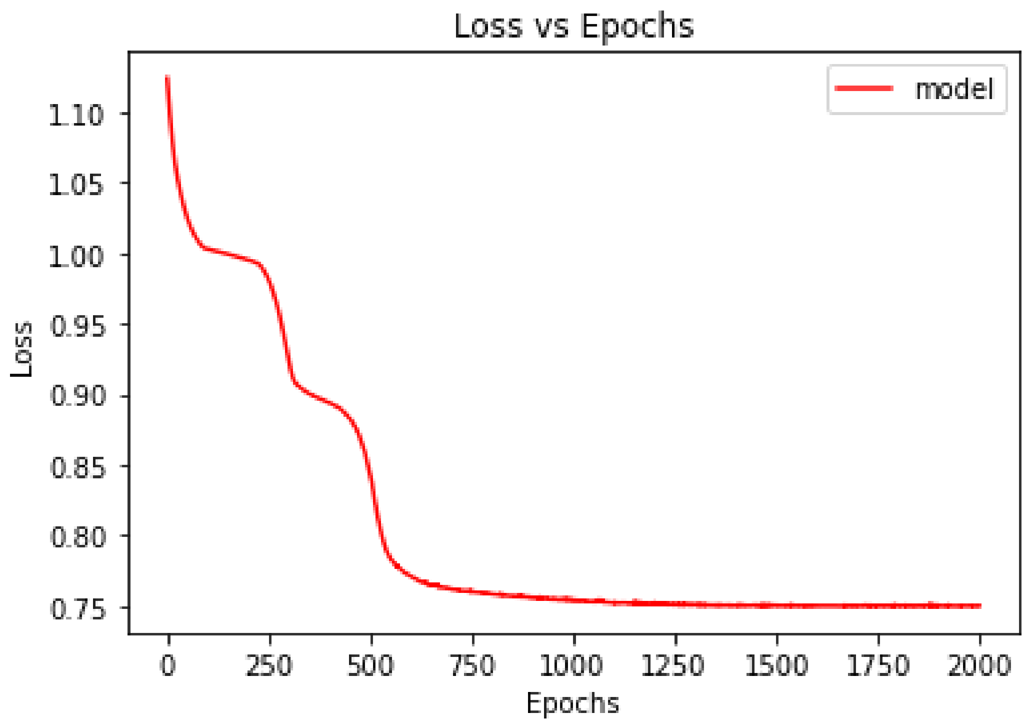
Task2: Observe Gradient Norm during Training

The model underwent training for multiple epochs, and the image below illustrates the gradient norm for each epoch.



Course: 8430 (Deep Learning)

The model lost data for each epochs are attached below



Result: The training and convergence of these models occur over time. We notice a slight increase in the gradient at 250 epochs, and simultaneously, there's a change in the slope of the loss. This is illustrated in the image above.

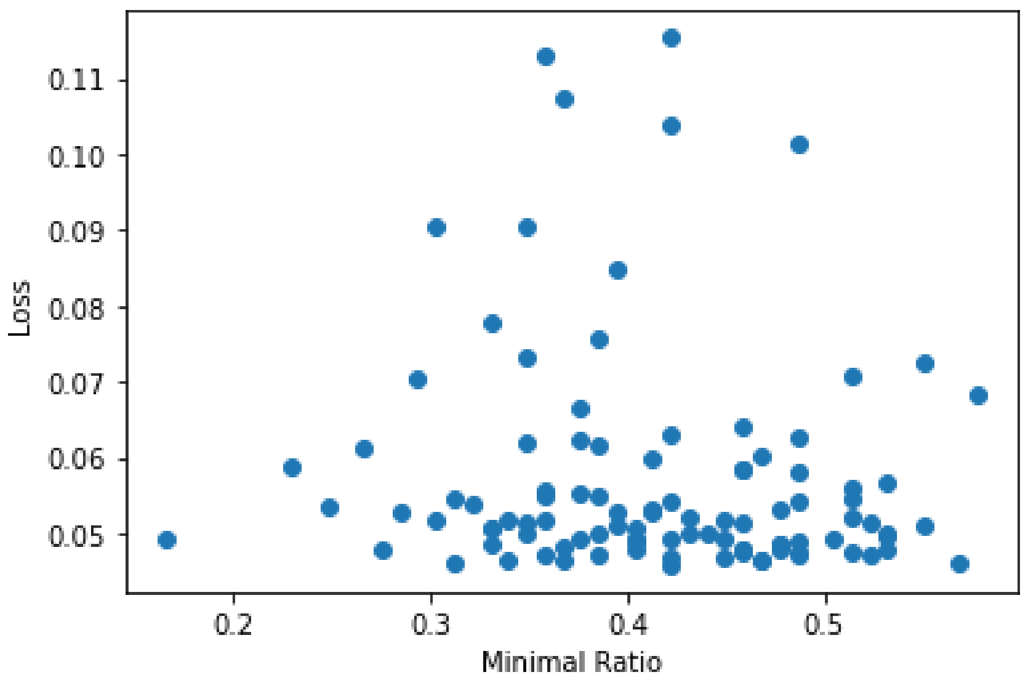
Task3: What happens when gradient is almost zero?

When the gradient of a function is near zero, it suggests that there is minimal change in that direction, indicating a flat slope at that specific point. A gradient close to zero signifies that the algorithm is approaching the function's minimum or maximum value, making this concept crucial for optimization algorithms like gradient descent.

To train the model, we used a total of one hundred epochs and employed the Adam optimizer with a learning rate of 0.001. To find the total number of parameters in the neural network, we need to do some calculations. The network consists of two hidden layers, each containing 32 parameters.

Course: 8430 (Deep Learning)

Result:



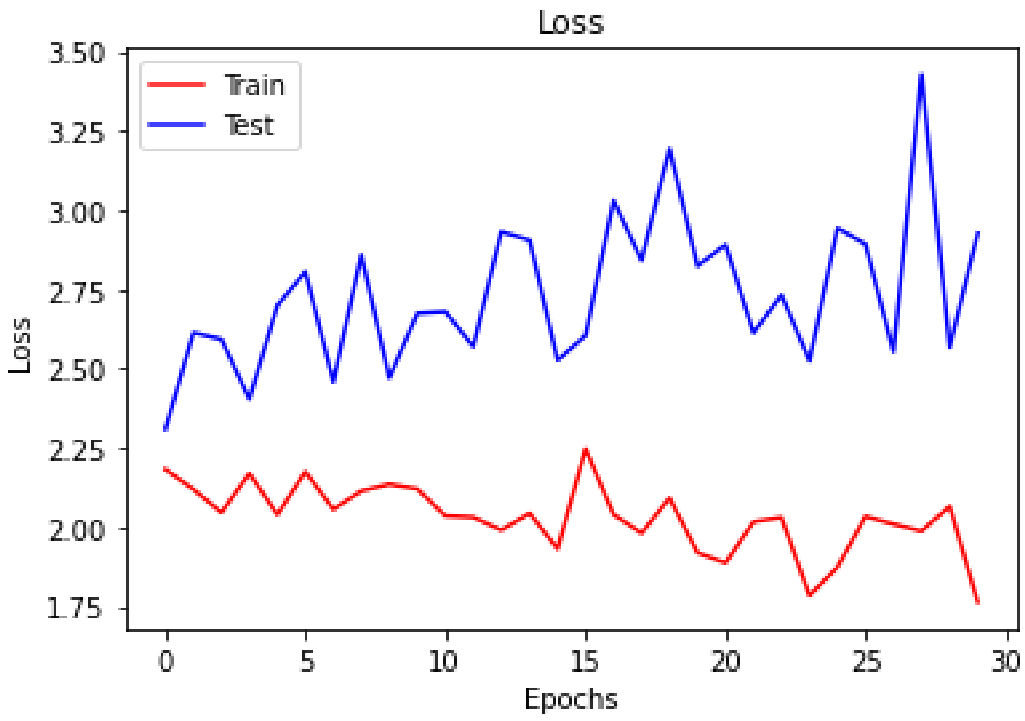
HW1-3: Generalization

Task1: Can network fit random labels?

The models were trained on the MNIST dataset, which consists of a training set of 60,000 images and a test set of 10,000 images. The training set was specifically used for model training. A learning rate of 0.001 was applied. To build the deep neural network, which included three hidden layers, a feedforward approach was employed. For the optimization process, I used the Adam optimizer.

The image displayed below was created using various random models. To reduce losses, the model needs to learn from these random labels and try to retain them as we move through the epochs. This makes the training process quite difficult, as it demands that the model learns from these random labels.

Course: 8430 (Deep Learning)



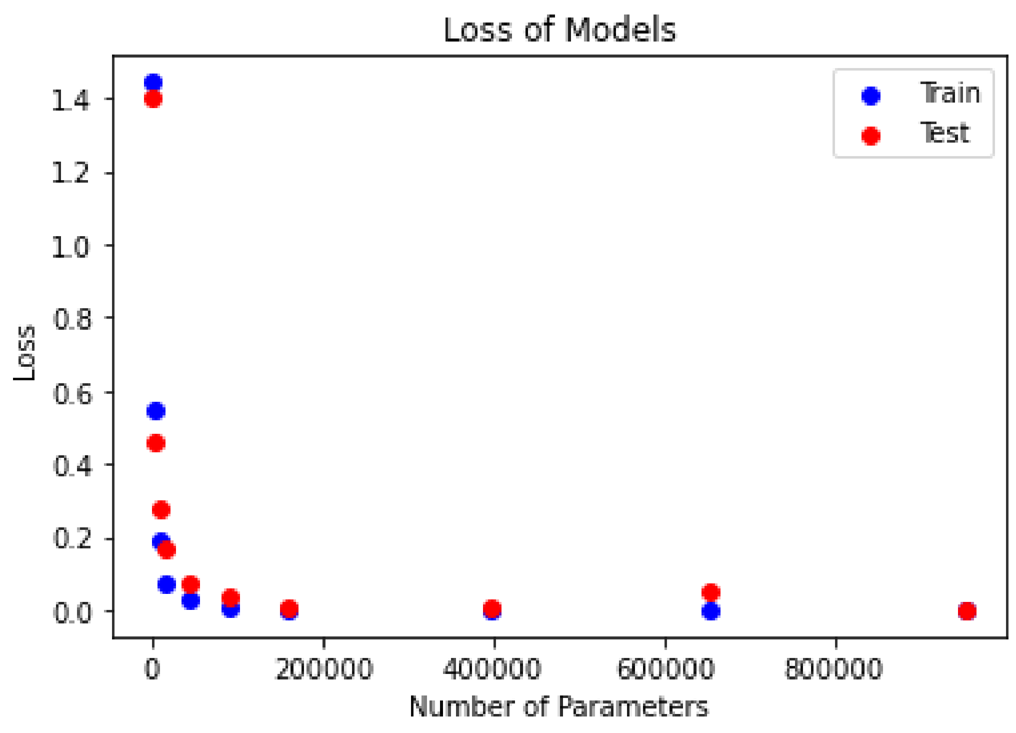
In the image shown above, we observe that the training loss is relatively low, while the testing loss is considerably higher. This indicates that the model struggles to fit random labels effectively.

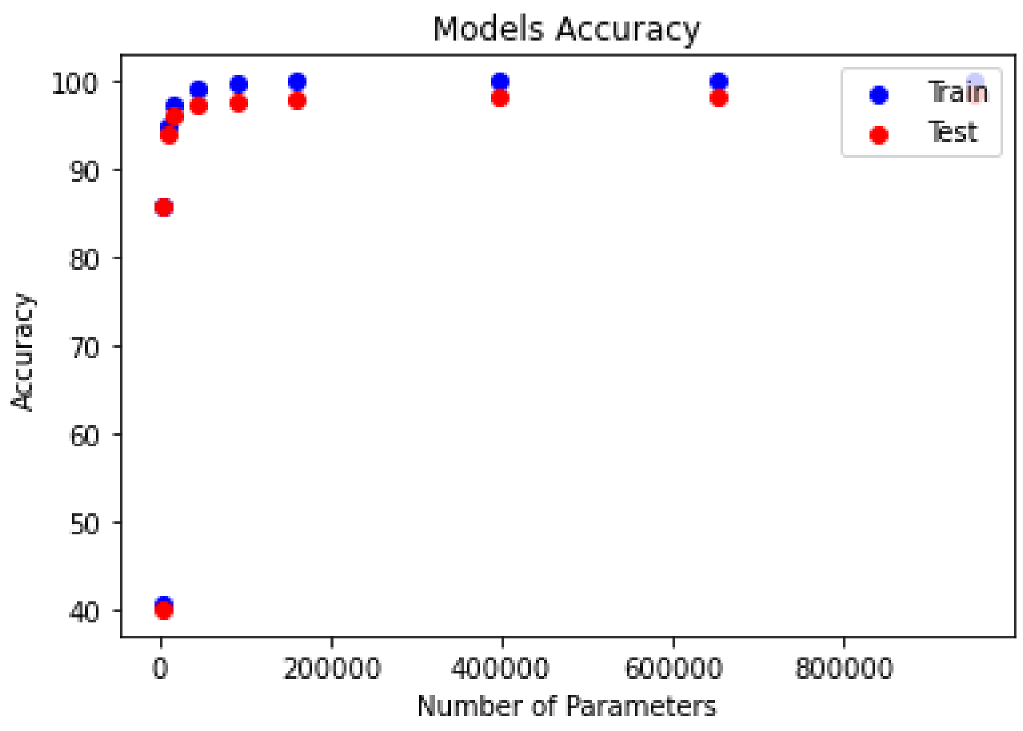
Task2: parameters v.s. Generalization

The MNIST dataset includes a training set of 60,000 samples and a test set of 10,000 samples, which the models are trained on throughout their training. For mesh optimization, I utilized the Adam optimizer and set the learning rate to 0.001. There may be hundreds of thousands or millions of parameters, depending on the model.

As you can see from the graphs below, the difference in loss and accuracy between training and testing grows as the number of parameters grows. This trend is further supported by the figures shown below. Once a certain threshold of parameters is achieved, the model's improvement from one iteration to the next is minimal. This suggests that adding more parameters provides little additional benefit to the model.

Course: 8430 (Deep Learning)





Using the training dataset instead of the test dataset results in better accuracy and lower model loss values.

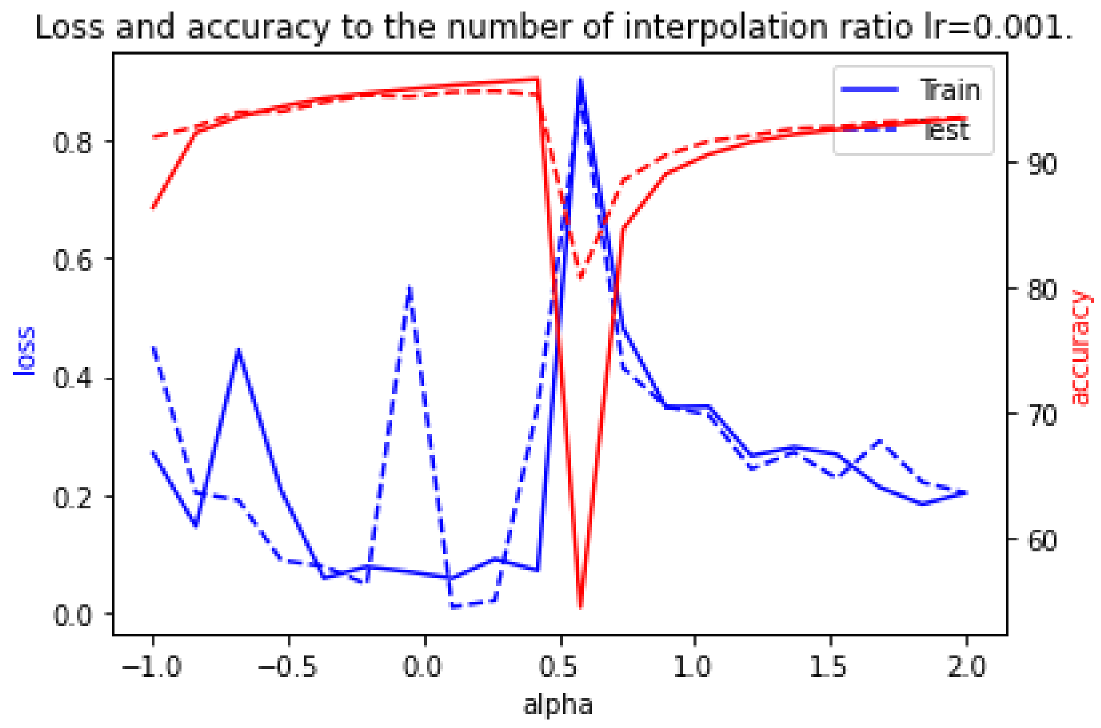
Task3: Flatness v.s. Generalization

Course: 8430 (Deep Learning)

Flatness v.s. Generalization Part1:

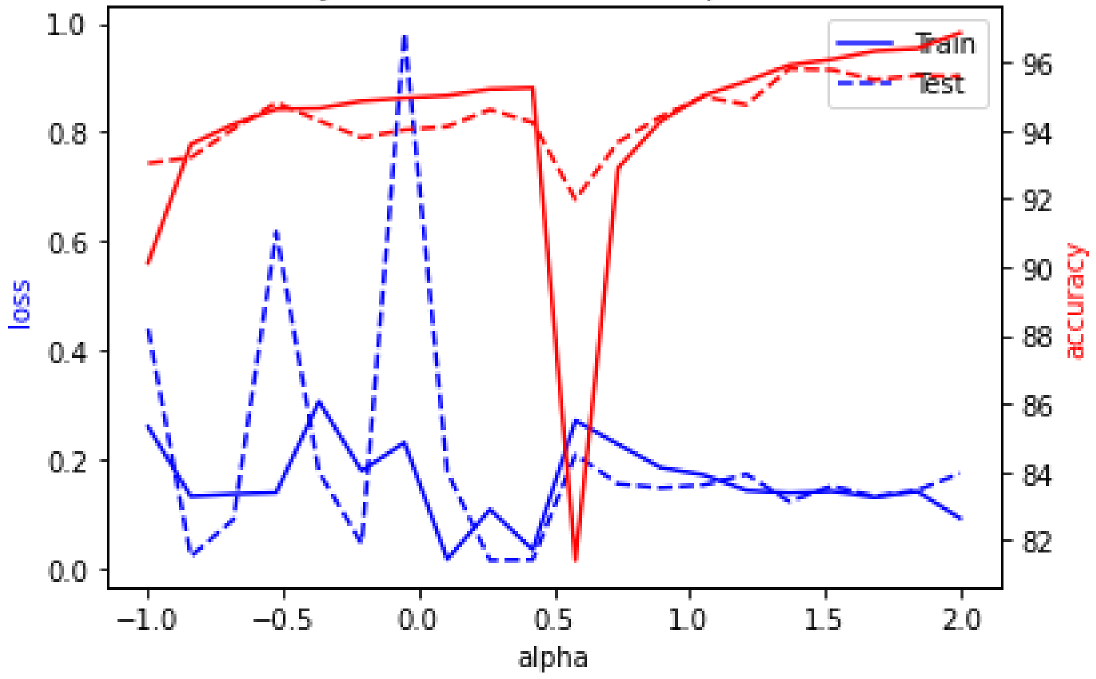
An MNIST dataset, comprising a training set of 60,000 images and a test set of 10,000 images, is utilized to train the model's capabilities. undergoes two training sessions with the same hyperparameters but different dose sizes. A batch size of 1024 is used in the first session, while a batch size of 64 is used in the second session. This method allows comparison of model performance and learning dynamics with different batch sizes.

After completing the training of the model with different batch sizes, we will gather the weights P1 and P2. Next, we will apply the formula Pi = (1 - alpha) * P1 + alpha * P2 to perform the interpolation calculation. To assess training loss, test loss, test accuracy, and training accuracy, we will proceed to train and test on new data using these updated weights.



Course: 8430 (Deep Learning)

Loss and accuracy to the number of interpolation ratio lr=0.01.

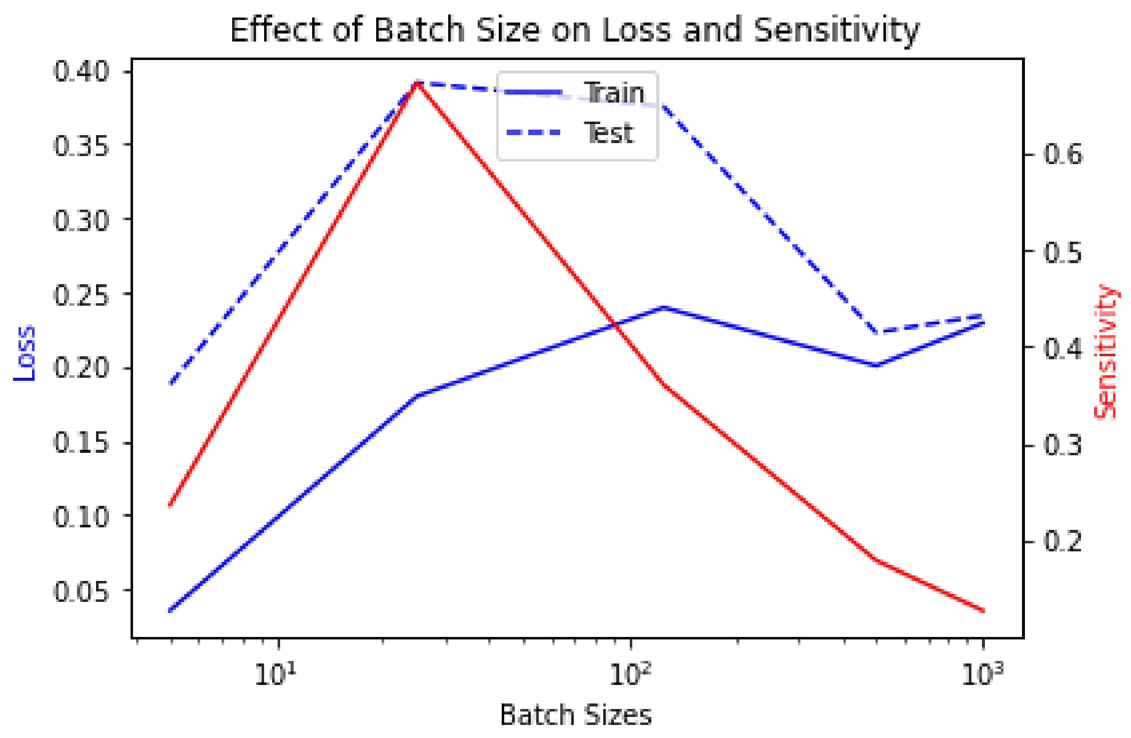


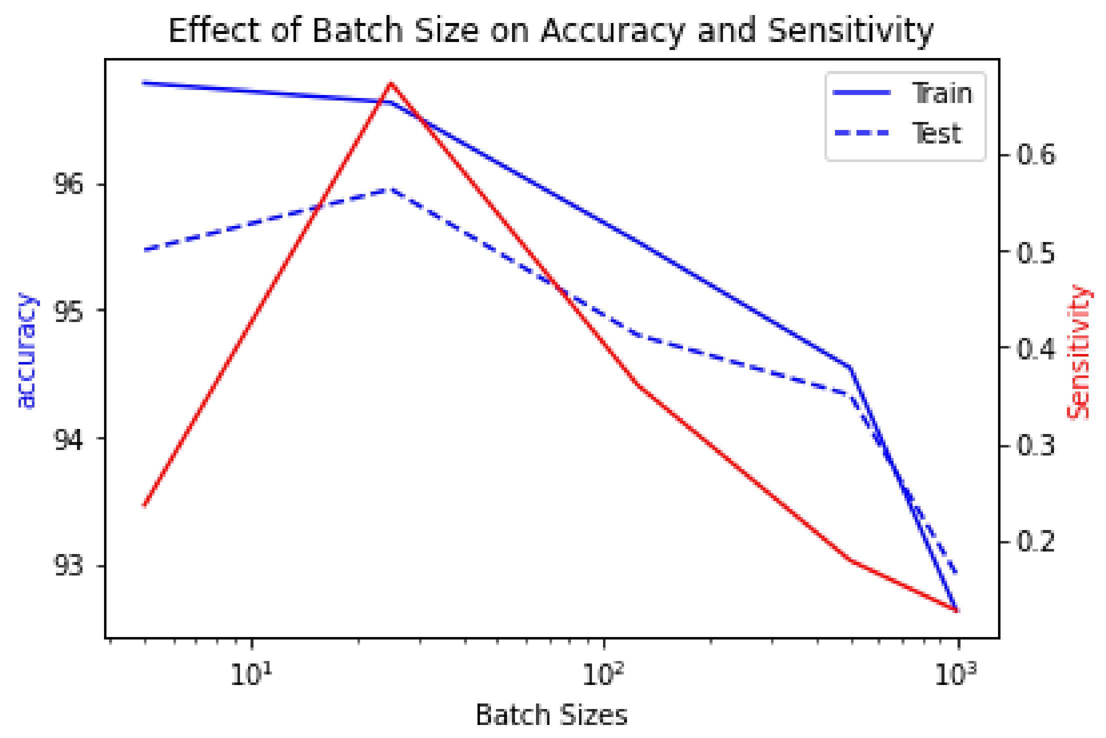
Flatness v.s. Generalization Part2:

Five identical Deep Neural Networks (DNNs), each with two hidden layers, were trained using the MNIST dataset, which includes 60,000 training samples and 10,000 testing samples. The Adam optimizer was utilized with a learning rate of 0.001 to improve the networks' performance. Various batch sizes, ranging from 5 to 1000, were tested to evaluate their impact on the models. After training, the accuracy and loss for both the training and test datasets were analyzed across the five models. Furthermore, the sensitivity of each model was examined using the Frobenius norm of the gradient method, highlighting the relationship between batch size and model performance.

The results shown in the accompanying images illustrate how batch size affects loss, accuracy, and sensitivity. Larger batch sizes demonstrated a distinct trend in sensitivity and accuracy, while smaller batch sizes affected model stability in a more varied way. This analysis clarifies how different batch sizes influence both the accuracy and sensitivity of the models, providing important insights for optimizing neural network training by considering batch size as a crucial hyperparameter.

Course: 8430 (Deep Learning)





Result: The images shown above indicate that optimal results are achieved with batch sizes ranging from 10^1 to 10^3, highlighting training and testing accuracy, loss, and sensitivity as the key factors.