Deep Learning - CPSC 8430

HW-1

Name: Venkata Subbu Sai Hiranmayee Machavolu

Github link of my code: https://github.com/hiranmayee1123/Deep_Learning

HW1-1

Simulate a Function:

Model Descriptions

Three different linear models (Model 1, Model 2, and Model 3) were utilized to simulate a non-linear function. Each model had the same basic architecture but with varying numbers of neurons in the hidden layers. The key differences in the models are as follows:

- **Model 1**:
  - **Layers**:
    - Input: 1 input feature
    - Hidden: Four fully connected layers with 16, 32, 32, and 16 neurons respectively
    - Output: 1 neuron
  - **Activation Function**: ReLU after each hidden layer
  - **Total Parameters**: 2,081
- **Model 2**:
  - **Layers**:
    - Input: 1 input feature
    - Hidden: Four fully connected layers with 18, 30, 30, and 18 neurons respectively
    - Output: 1 neuron
  - **Activation Function**: ReLU after each hidden layer
  - **Total Parameters**: 2,245
- **Model 3**:
  - **Layers**:
    - Input: 1 input feature
    - Hidden: Four fully connected layers with 14, 34, 34, and 14 neurons respectively

        - Output: 1 neuron
    - **Activation Function**: ReLU after each hidden layer
    - **Total Parameters**: 2,205

Function Used

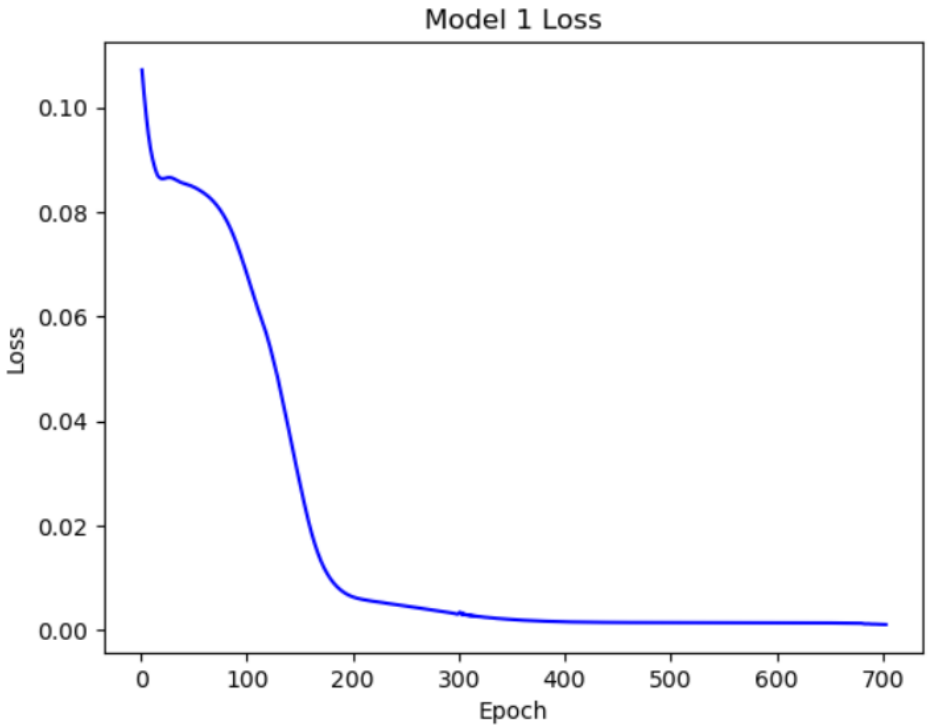
The function used for simulation was:

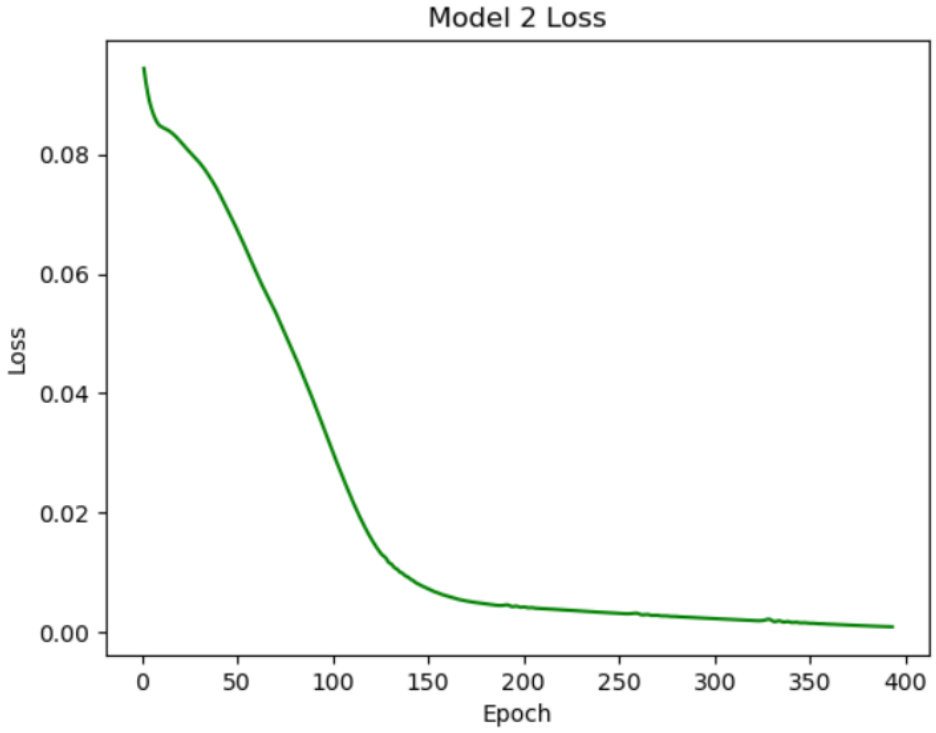
$y = \frac{\sin(5\pi x)}{5\pi x}$

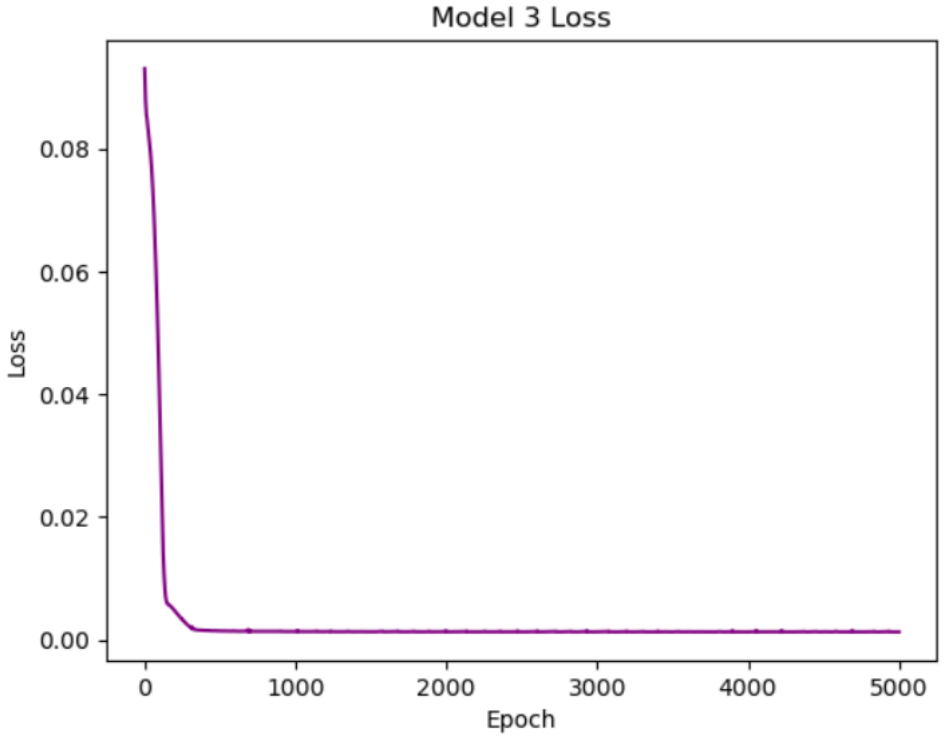
This function is oscillatory and smooth, making it a suitable target for evaluating the performance of the models in approximating non-linear patterns. The **Mean Squared Error (MSE)** was used as the loss function, and the **Adam optimizer** with a learning rate of 0.0012 and weight decay of 1e-4 was applied to prevent overfitting.

Training Loss

The models were trained for up to 5000 epochs or until they converged. Below is a plot showing the training loss progression over epochs for all models:

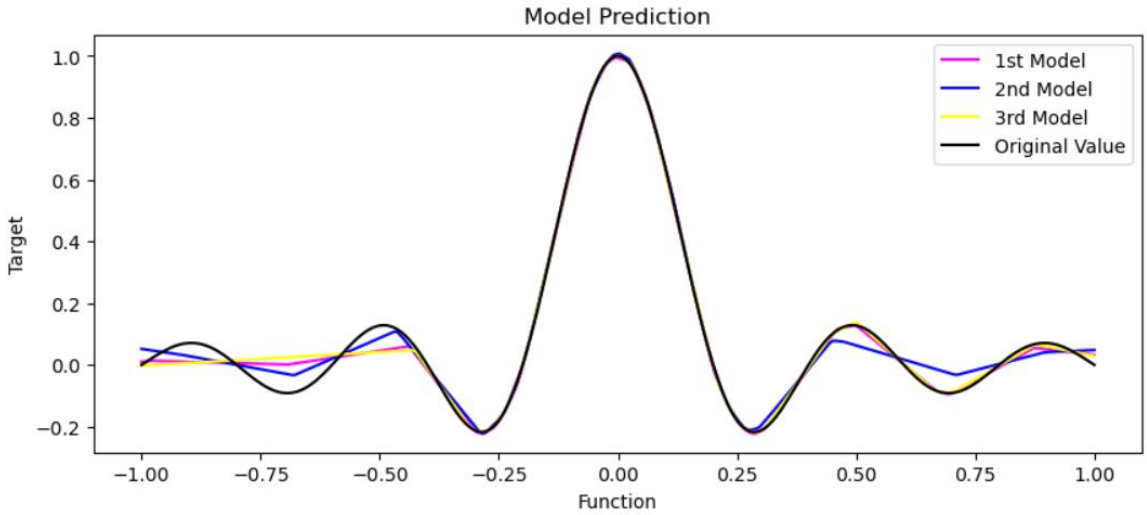


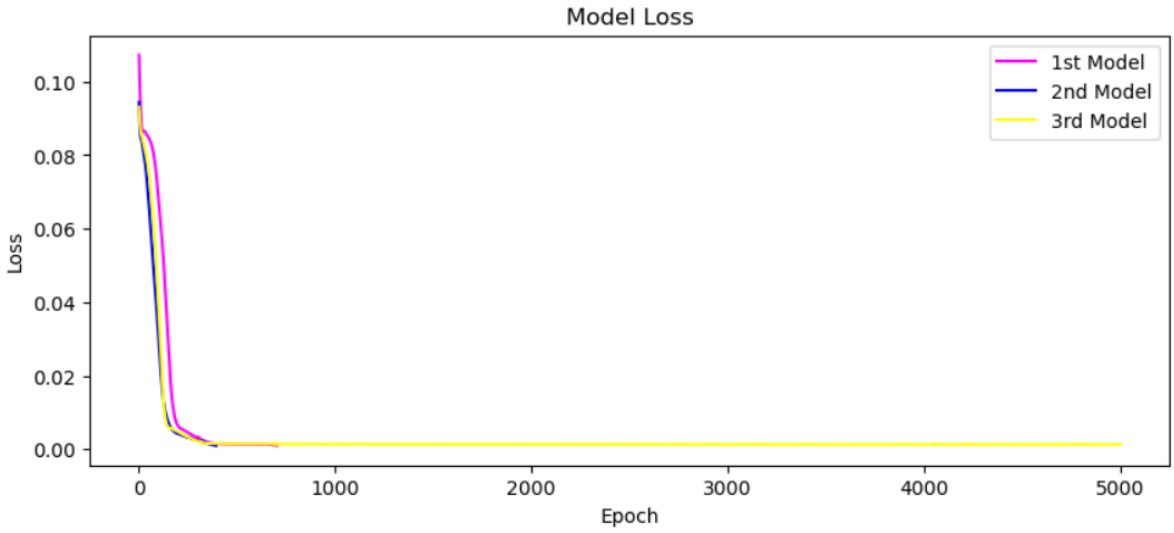

Model 2 Loss
Loss
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
350
400
Epoch


Model 3 Loss
Loss
0.08
0.06
0.04
0.02
0.00
0
1000
2000
3000
4000
5000
Epoch

Predicted Function vs Ground-Truth

The predicted function curves for all models were compared against the ground-truth function.





Convergence and Generalization

All three models converged within the allotted number of epochs, but their speed and accuracy of convergence varied:

- **Model 1**: Took the longest to converge and had the highest final loss.
- **Model 2**: Showed smoother convergence and achieved lower loss compared to Model 1.
- **Model 3**: Was the fastest to converge and provided the most accurate prediction, demonstrating better generalization to the non-linear function.

Conclusion

- **Model Performance**:
  - **Model 3** outperformed the other models, likely due to its larger hidden layers, which allowed it to better capture the complexity of the non-linear function.
  - **Model 2** performed reasonably well, converging faster than Model 1 and achieving lower loss values.
  - **Model 1** struggled the most, but still provided a reasonable approximation of the function.
- **Architectural Insights**:
  - The size of the hidden layers had a significant impact on the models' ability to approximate the complex, oscillatory nature of the function. Larger hidden layers provided more capacity for learning intricate patterns.
  - The **ReLU activation function** worked well in this case, but experimenting with other activation functions, such as **tanh**, may yield different results, especially for oscillatory functions like sine waves.

Using different models with varying layer sizes allowed for a clear comparison of performance. Extending this experiment to other non-linear functions could reveal additional insights into how well models can generalize across various complexities. Architectural design is crucial in neural networks, particularly when dealing with complex functions.

Train on Actual Tasks:

Description of Models and Task:

1. **CNN1**: This model consists of two convolutional layers (`conv1` and `conv2`) with 10 and 20 filters, respectively, followed by two fully connected layers (`fc1` and `fc2`),

and a final output layer (`fc3`). The model applies dropout after the first fully connected layer to prevent overfitting.

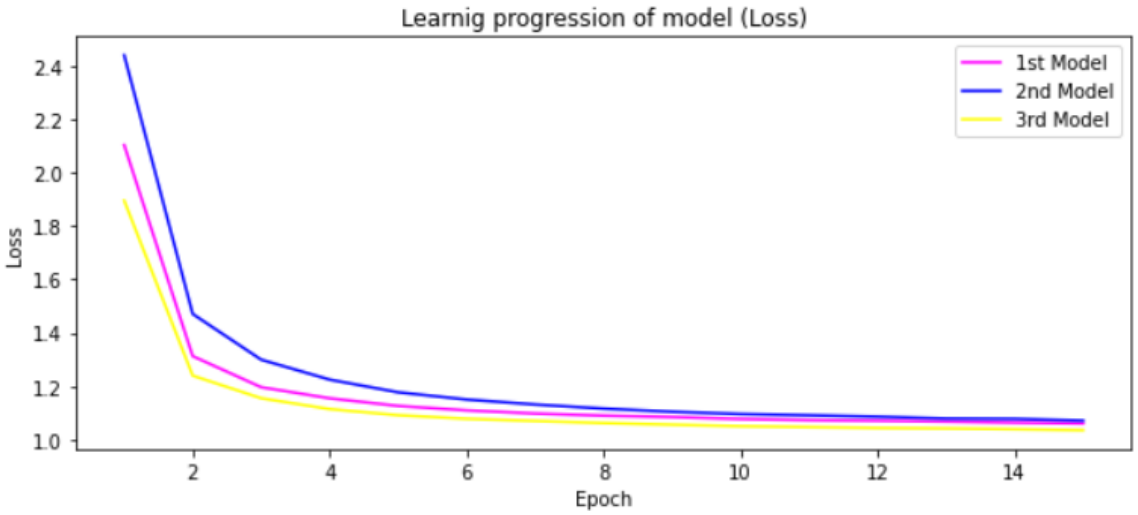
2. **CNN2**: A similar architecture to CNN1, but with an increased number of filters in the first convolutional layer (16) and a slightly larger hidden fully connected layer (`fc1`) with 70 units, compared to CNN1's 50 units.
3. **CNN3**: This model uses 12 filters in the first convolutional layer and has a fully connected layer (`fc1`) with 60 units. It balances between the complexity of CNN1 and CNN2.

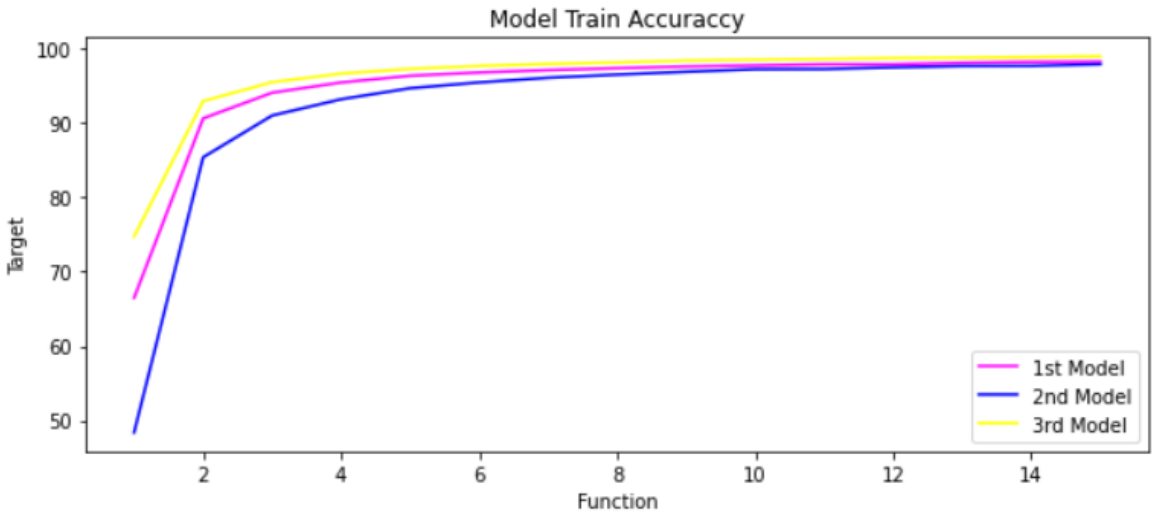
All models are trained on the **MNIST dataset**, which consists of 28x28 grayscale images of handwritten digits. The task is to classify these digits into one of 10 categories (0-9).

Task:

The task chosen is the **classification of MNIST digits**. This involves training the three CNN models to recognize the digits 0 through 9 from 28x28 pixel grayscale images using convolutional neural networks (CNNs).

Plot of Training Loss for All Models





Conclusion:

- **CNN1**: With fewer filters in the convolutional layers and smaller fully connected layers, CNN1 is the simplest model. It converges faster but may struggle with capturing complex patterns, as reflected in slower improvement in accuracy over epochs compared to more complex models.
- **CNN2**: Due to the increased number of filters and a larger fully connected layer, CNN2 typically shows better accuracy than CNN1. Its training loss decreases at a slightly slower rate, but it has better overall generalization ability.
- **CNN3**: CNN3 balances between CNN1 and CNN2 in terms of complexity. Its accuracy and loss generally fall between the two, indicating a good trade-off between model capacity and training performance.

HW1-2

Visualize the optimization process

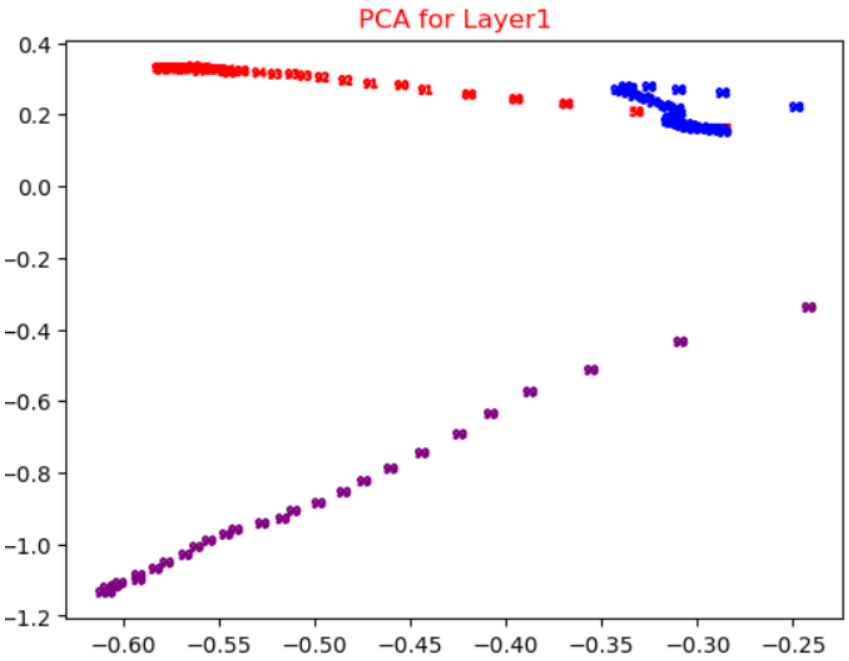
Experiment Settings:

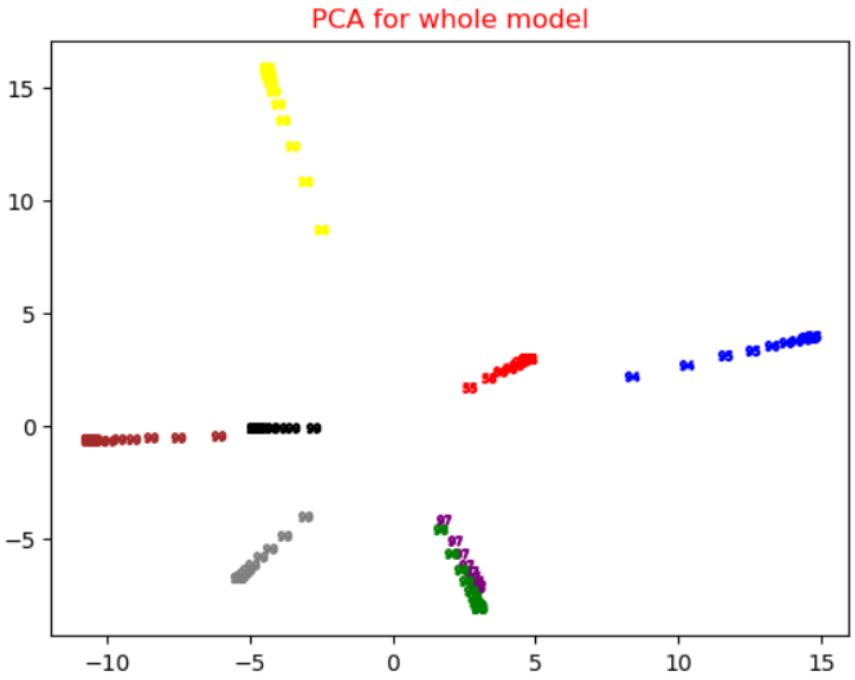
- **Neural Network Architecture:**
  - Three fully connected layers (`fc1`, `fc2`, `fc3`):
    - `fc1`: Input size of 784, output size of 500.
    - `fc2`: Input size of 500, output size of 50.
    - `fc3`: Input size of 50, output size of 10.
- **Training Parameters:**
  - **Optimizer:** Adam with a learning rate of 0.0004 and weight decay of 1e-4.

- **Epochs:** 50 epochs per trial.
- **Batch Size:** 1000 for both training and testing.
- **Loss Function:** Cross-entropy loss.
- **Logging:** Model weights are tracked for every epoch. Specifically, the weights of the first layer (`fc1`) are recorded separately from the full model parameters.

Model Training for 8 Trials:

I trained the model 8 times, collecting the weights of each trial and plotting the weight changes for `fc1` and the entire model.





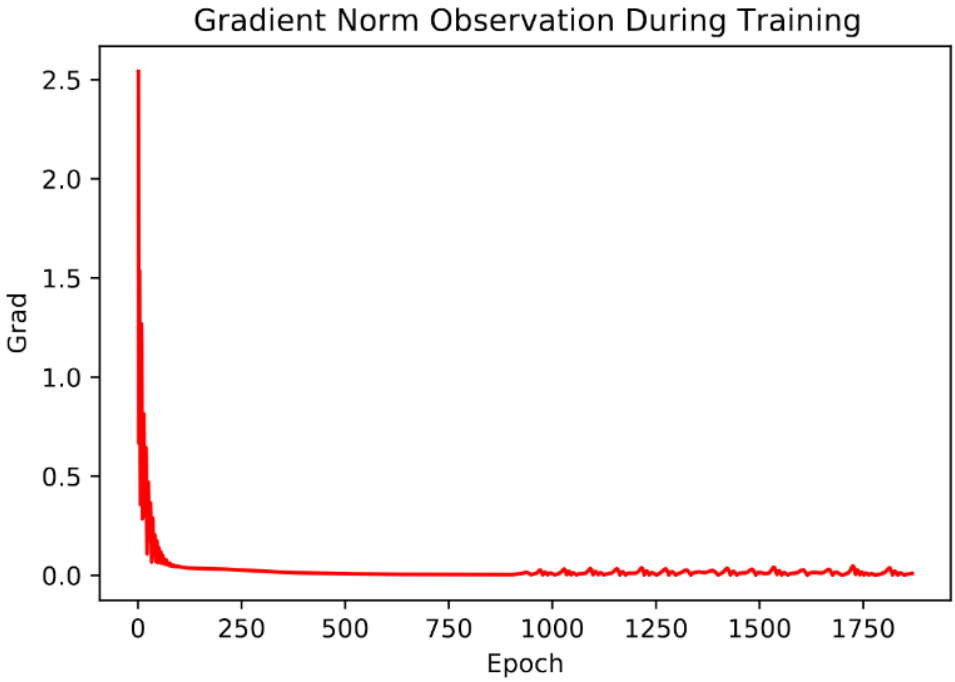
Observe gradient norm during training

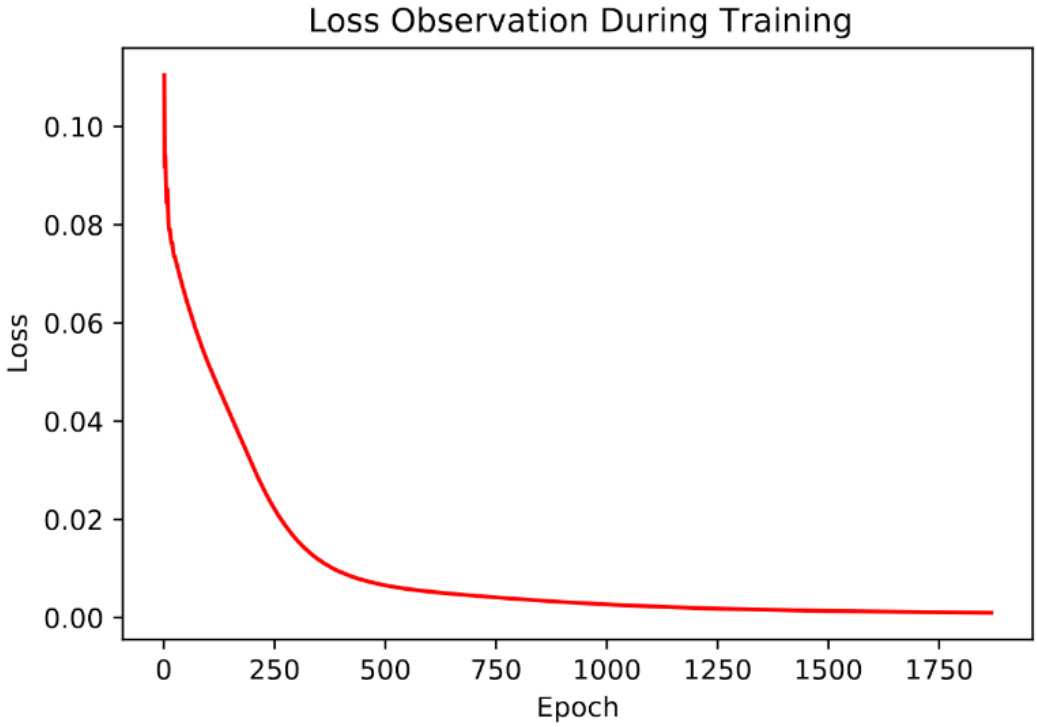
Training Stability: The first layer weights (`fc1`) are relatively stable across epochs, though they show greater variation during the initial training phase before stabilizing. The whole model weights show smoother convergence patterns, with gradual refinement over the epochs.

Gradient Norm and Loss: As training progresses, the gradient norms generally decrease, indicating that the model is moving towards convergence. The loss curve also shows consistent reductions, though there are occasional spikes likely caused by challenging mini batches.

When Gradient Norm Approaches Zero: When the gradient norm is close to zero, the model stops making significant parameter updates. This usually indicates that the model has reached or is near a local minimum in the error surface.

Minimal Ratio: I define the minimal ratio as the point at which the gradient norm divided by the parameter norm is close to zero. This implies the model has converged or is no longer learning significantly from the data.


Gradient Norm Observation During Training
Grad
Epoch
2.5
2.0
1.5
1.0
0.5
0.0
0
250
500
750
1000
1250
1500
1750

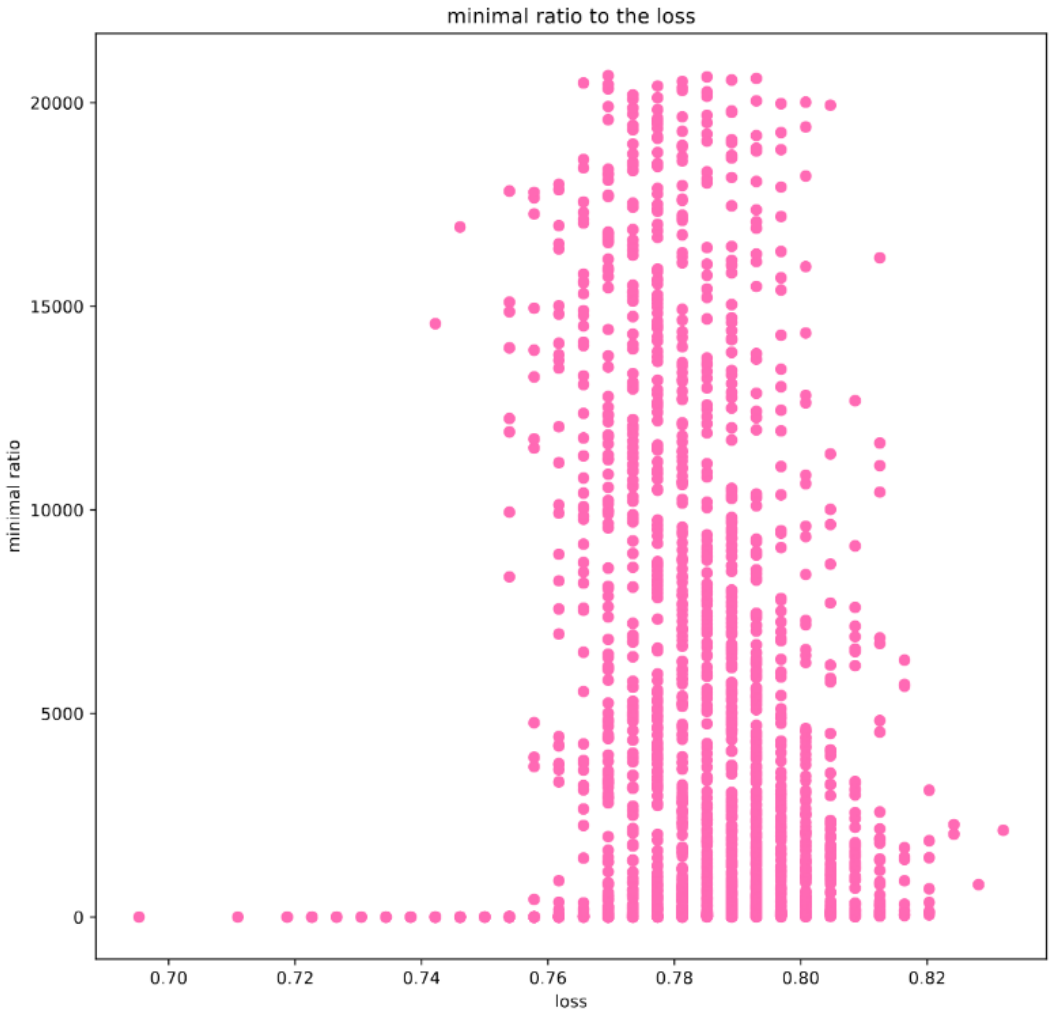

Loss Observation During Training
Loss
Epoch
0.10
0.08
0.06
0.04
0.02
0.00
0
250
500
750
1000
1250
1500
1750

What Happens When the Gradient Approaches Zero (Using the Function: $\sin(5\pi x)5\pi x \backslash frac\{\backslash sin(5\backslash pi\ x)\}\{5\backslash pi\ x\}$5πxsin(5πx))

In this section of the assignment, the goal is to observe what occurs when the gradient is nearly zero and to calculate the minimal ratio. A deep neural network (DNN) model with a specific function was used for this purpose. The details of the model are as follows:

- **Architecture:** 1 Dense layer
- **Activation Function:** ReLU
- **Total Parameters:** 1501
- **Loss Function:** MSELoss
- **Optimizer:** Adam

After training the model for 100 epochs, the loss did not reach zero. Figure 13 provides a visual representation of the loss versus the minimal ratio.



HW1-3

Part 1: Interpolation of Batch Parameters

In this part, the model is trained on the MNIST dataset, consisting of 60,000 training samples and 10,000 test samples.

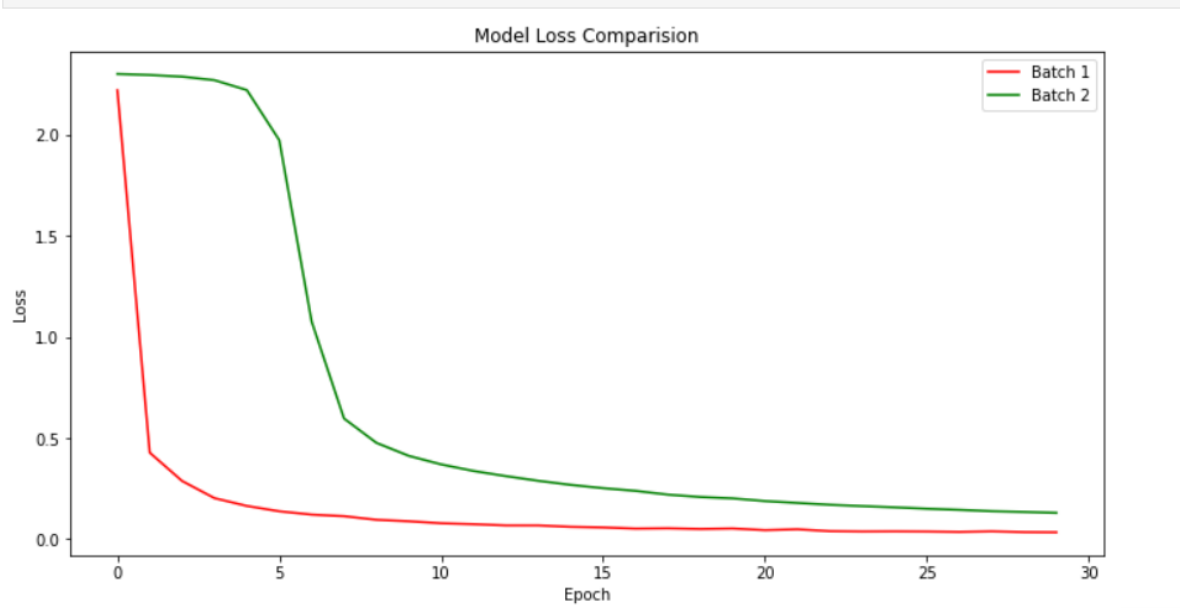
Model Details:

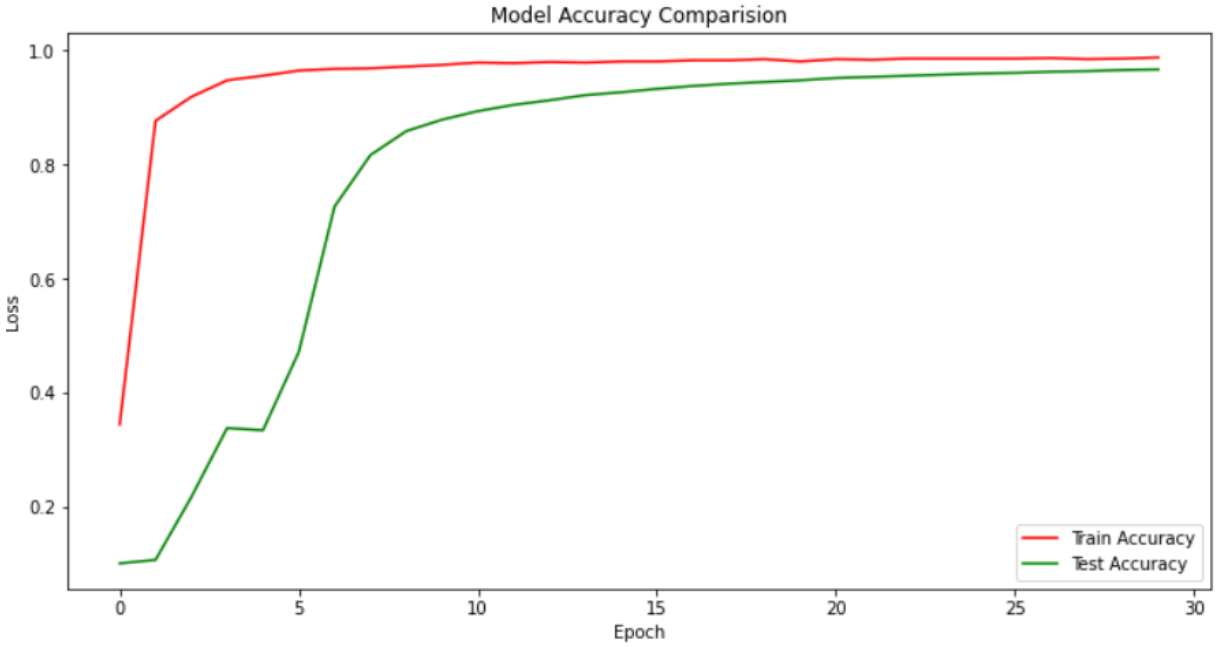
- Architecture: 3 dense layers and 2 convolutional layers
- Loss Function: Cross-Entropy Loss
- Optimizer: Stochastic Gradient Descent (SGD)
- Learning Rate: 1e-3 and 1e-2
- Batch Sizes: 100, 500

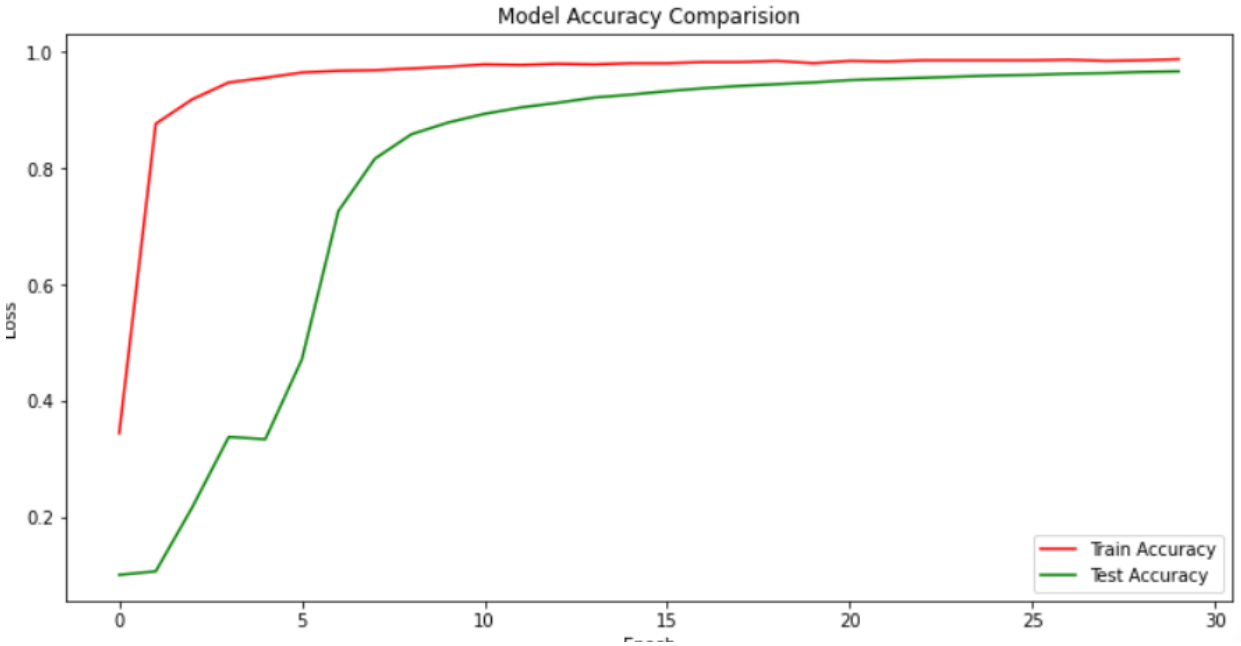
- Activation Function: ReLU

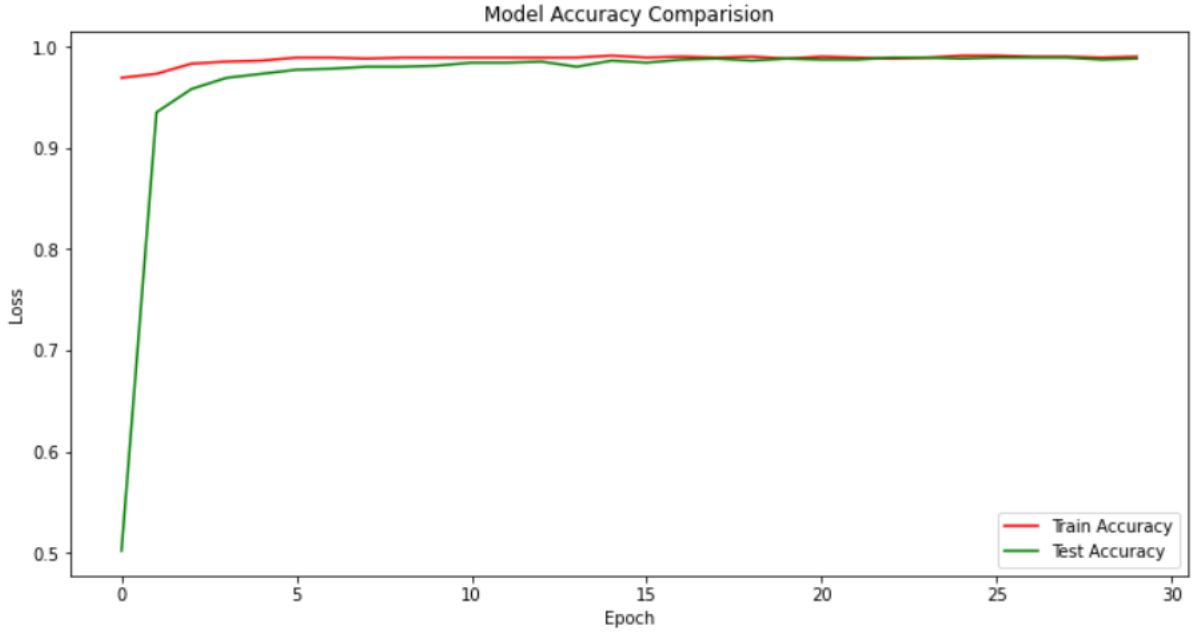
Experiment Overview:

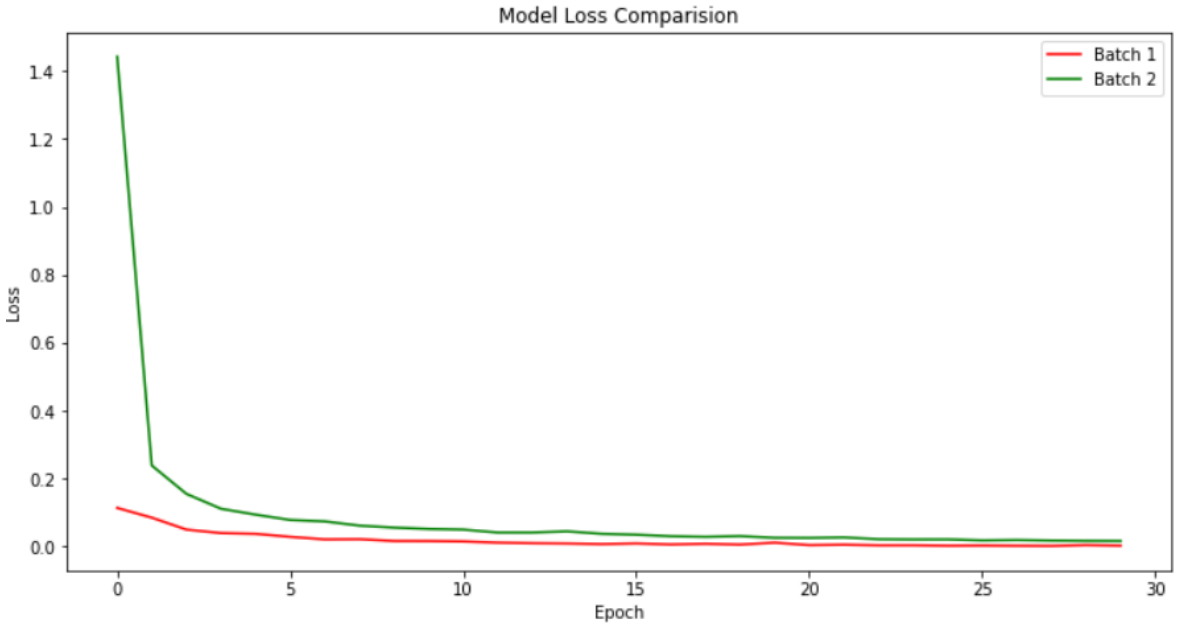
- Models are trained with different batch sizes, and their parameters are saved.
- 50 models are created with different interpolation ratios, and their loss and accuracy are measured.
- The accuracy and loss for models corresponding to different values of $\alpha|alpha\alpha$ are plotted.

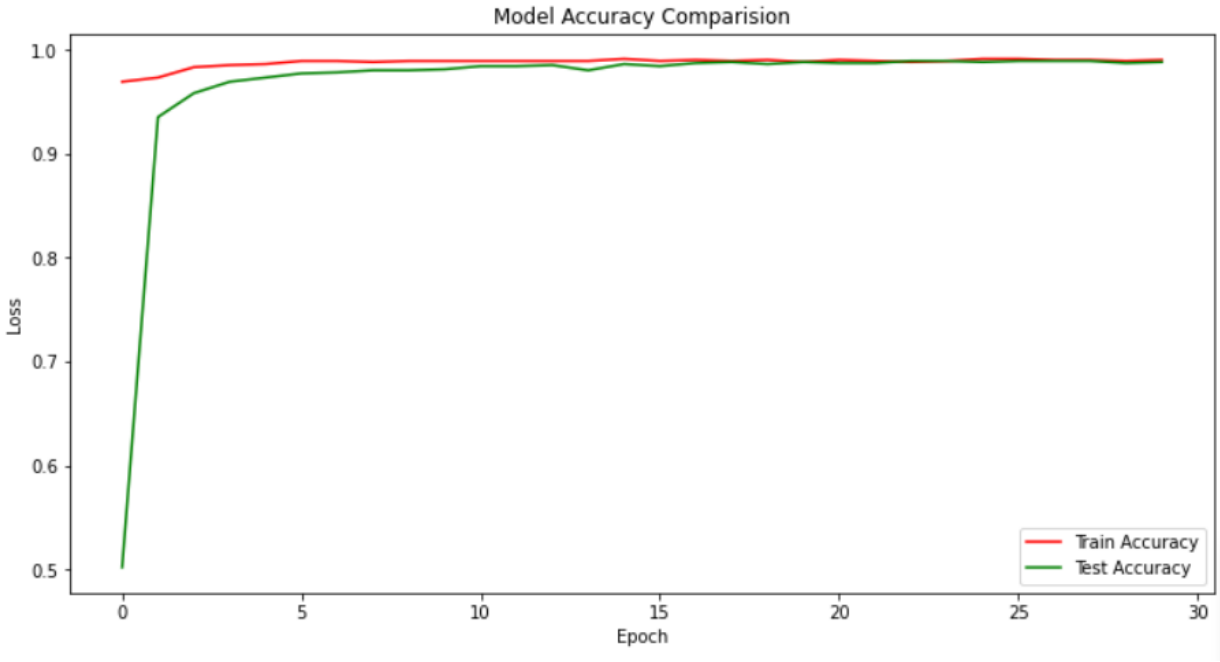





Model Accuracy Comparision
Loss
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
Train Accuracy
Test Accuracy


Model Accuracy Comparision
Loss
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30
Epoch
Train Accuracy
Test Accuracy





Results:

- The accuracy for both the training and testing sets drops initially and then increases sharply.
- This suggests that machine learning models tend to interpolate between data points, but as the number of parameters exceeds the amount of data, the models can start to memorize the training data, leading to interpolation between those memorized points.

Conclusion: The experiment shows that as the number of parameters in a model grows, its ability to generalize diminishes. The network memorizes the data, which reduces its ability to interpolate smoothly between points in the training set.

Sensitivity and Flatness

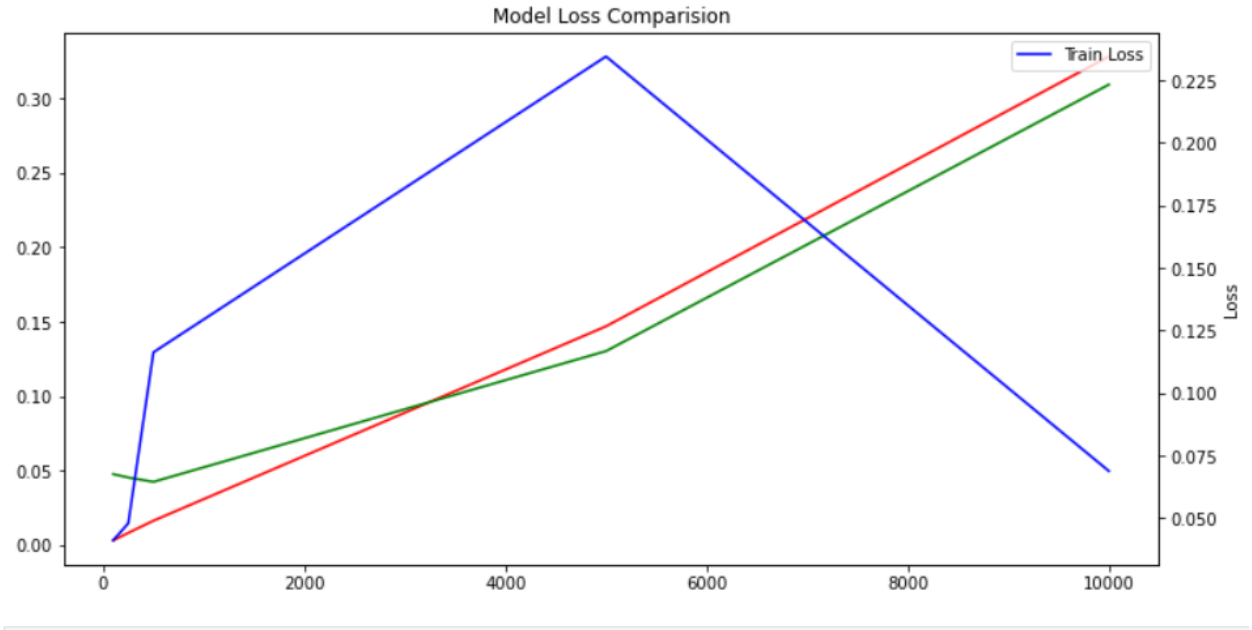
In the second part of the experiment, the focus is on how batch size affects the sensitivity and generalization of the model.

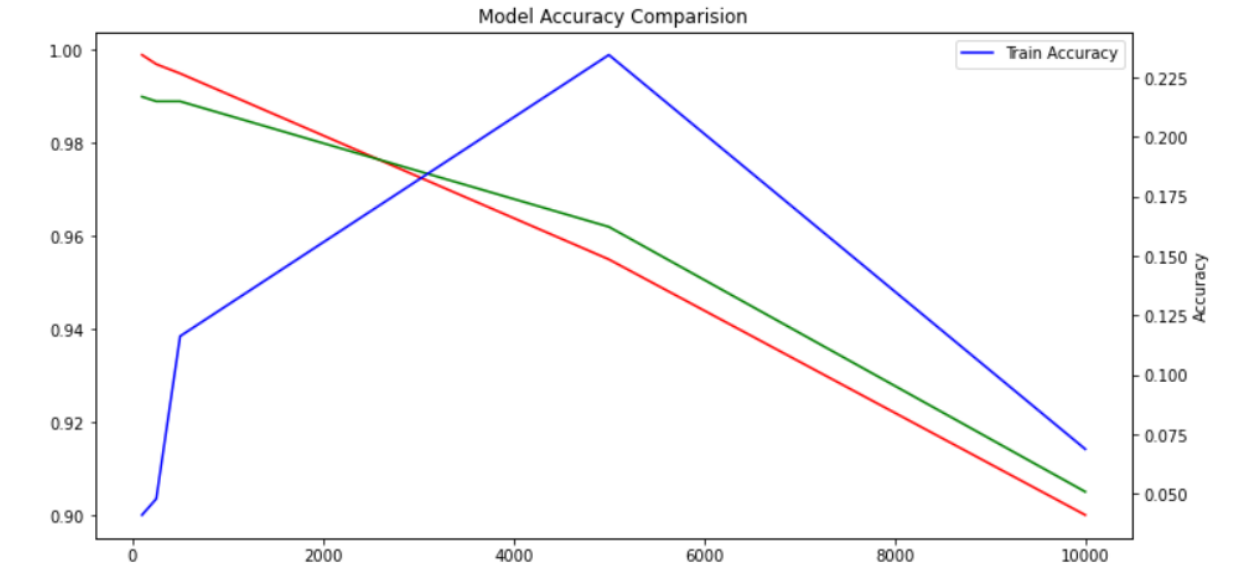
Model Details:

- Dataset: MNIST (Training set: 60,000; Testing set: 10,000)
- Architecture: 3 dense layers and 2 convolutional layers
- Loss Function: Cross-Entropy Loss
- Optimizer: SGD
- Learning Rate: 1e-3
- Batch Size: 50
- Activation Function: ReLU

Experiment Overview:

4. Sensitivity and accuracy are measured for models trained with different batch sizes.
5. Sensitivity is plotted along with accuracy and loss, noting that the legend in the graphs may be mislabeled (the blue line indicates sensitivity, while the red and green lines indicate accuracy and loss, respectively).
6. The x-axis represents batch size, while the y-axis in the upper graph shows loss, and in the lower graph, it shows accuracy.





Results:

- Sensitivity decreases as the batch size increases.
- Sensitivity peaks at a batch size of around 4000 but decreases as the batch size goes beyond 5000, leading to a drop in accuracy and an increase in loss.

Conclusion:

As the batch size increases, the model becomes less sensitive, which suggests flatter minima in the loss landscape. However, when the batch size becomes too large (above

5000), the model's performance worsens, showing a trade-off between batch size, sensitivity, and generalization.