Homework-1

Course: Deep Learning 8430
Name: Pragathi Pendem
Email: ppendem@g.clemson.edu

GitHublink:https://github.com/Pragathibruno/DeepLearning8430

HomeWork 1(1) Function Simulation

Give a brief description of the models you employ, mentioning their parameters (at least two models), as well as the function you employ.

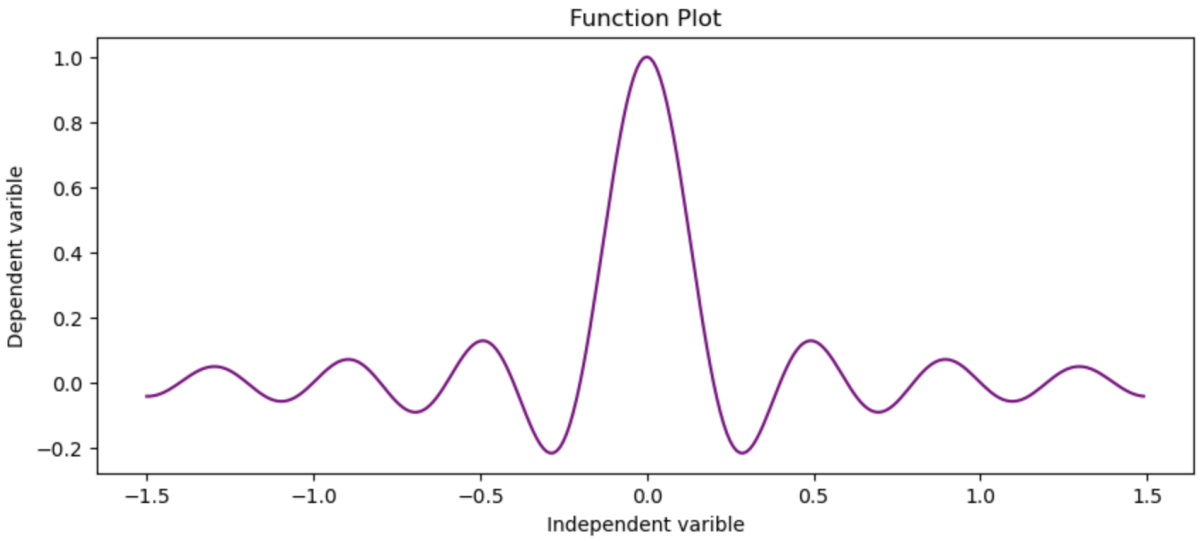
Three models I've created match the three shared by the requirements. By including a penalty term to a neural network's cost function, which shrinks the weights during backpropagation, a weight decay parameter has been added to help regularize the models more effectively.

Model 1: Has 571 parameters and seven thick layers. The optimizer is RMSProp, the learning rate is 1e-3, and the loss function is MSELoss. 1e-4 is the weight decay setting., and the activation function is LeakyRelu.

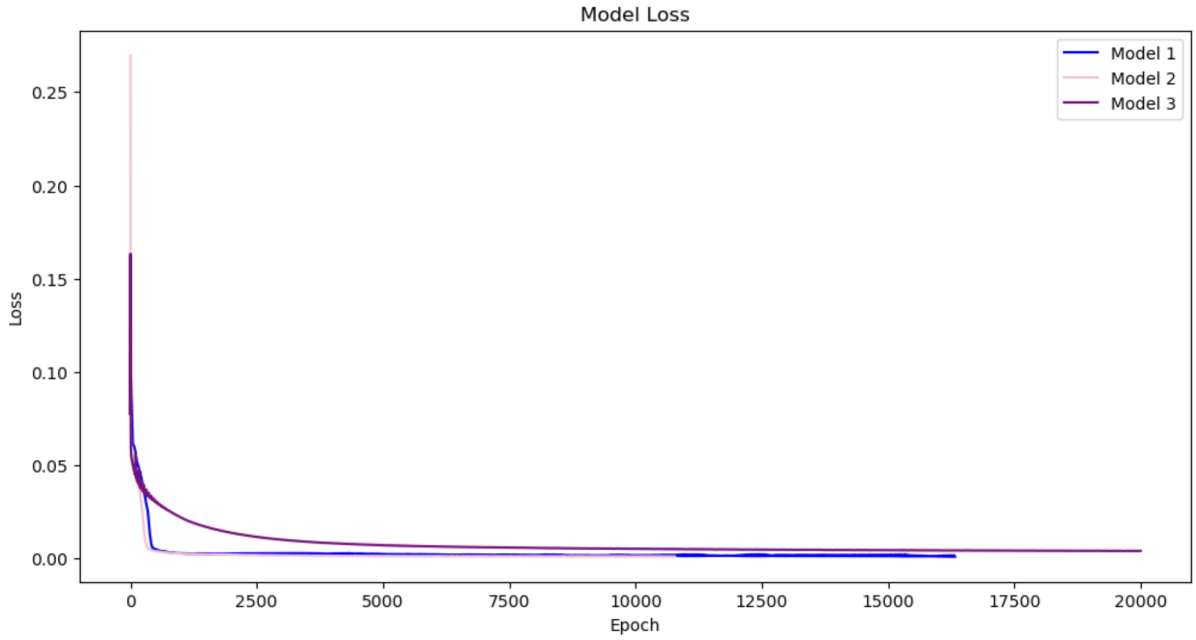
Model 2: Includes 572 parameters and four dense layers. The optimizer is RMSProp, the loss function is MSELoss, 1e-4 is the weight decay setting. LeakyRelu is the chosen activation function, and the weight decay is set to 1e-4.One dense layer with 571 parameters makes up

Model 3: The optimizer is RMSProp, the loss function is MSELoss, and the learning rate is 1e-3. 1e-4 is the weight decay setting., and the activation function is LeakyRelu.

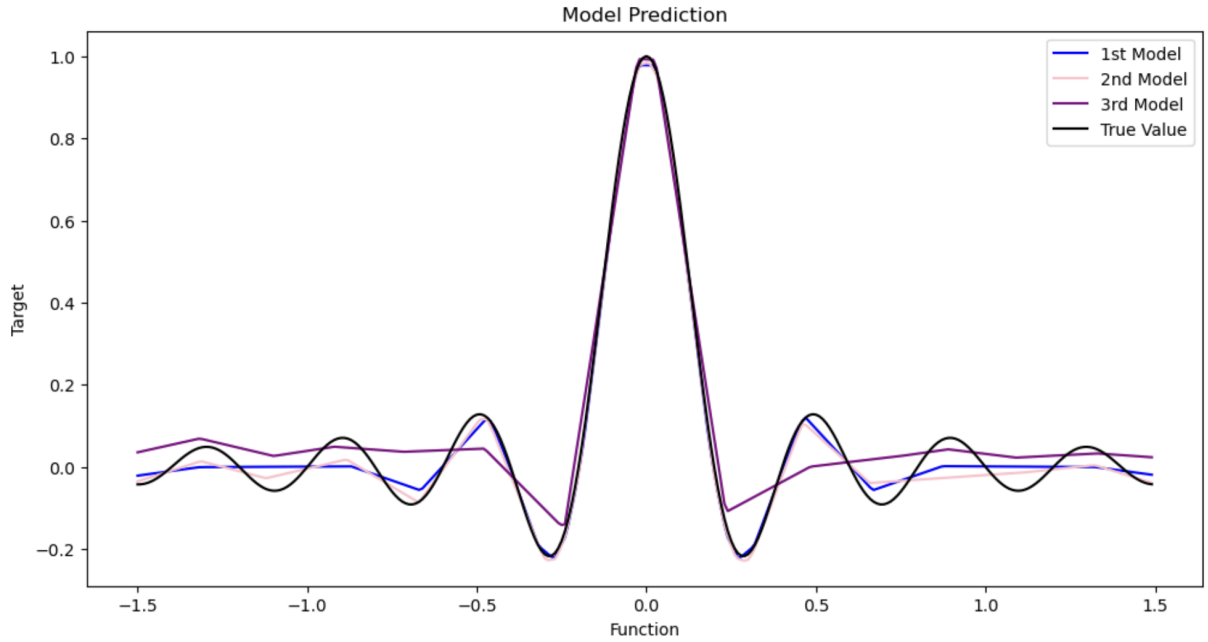
Function1: "sin(5pix) / 5pix" is the definition of the first function. The accompanying figure displays the function's plot.



Simulating the Function: All models converge once the model has run for the maximum number of epochs or when learning becomes sluggish. The loss for each of these models is shown in the graph below.

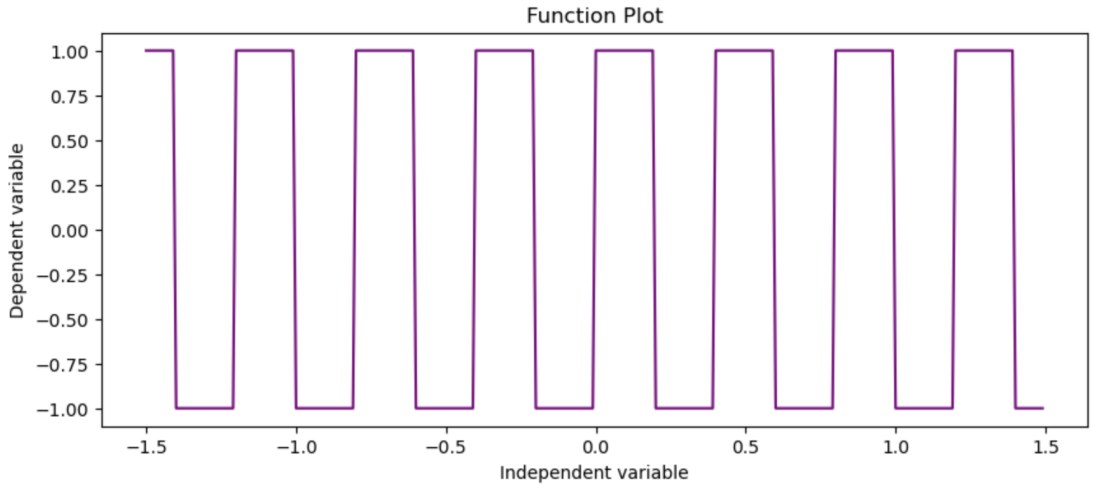


The graph below shows the ground reality for the three models compared to their forecasts.

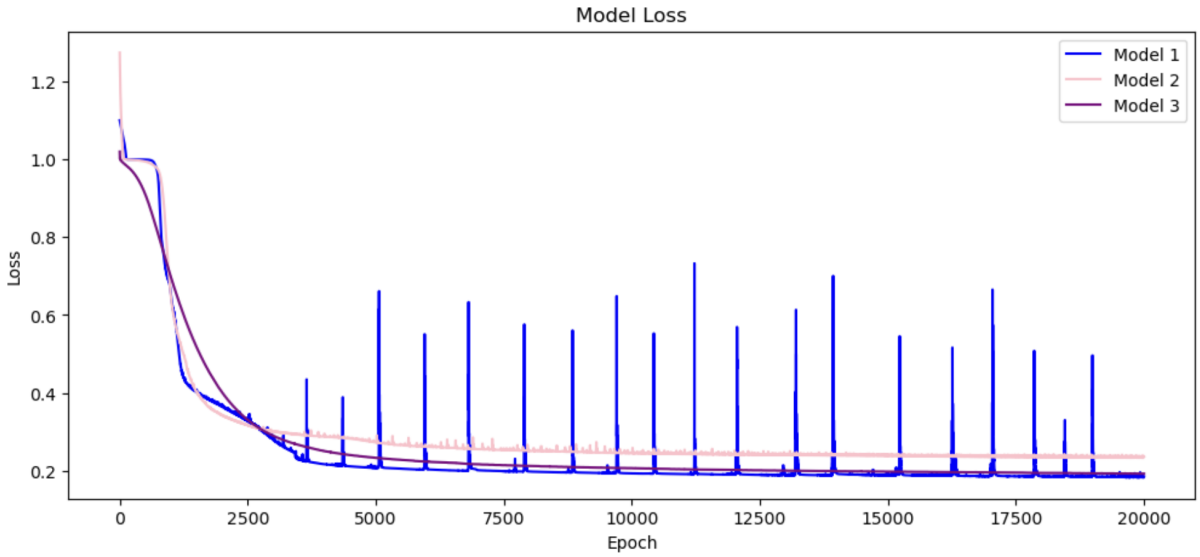


Result:
Models 1 and 2 converge more quickly than model 3, which approaches its maximum number of epochs before converging. Models 1 and 2 learn the function significantly more well than Model 3 and have lower loss values, as seen by the graph. This is evidence of the various models' layer counts, which allowed the models with more layers to learn more quickly and effectively.

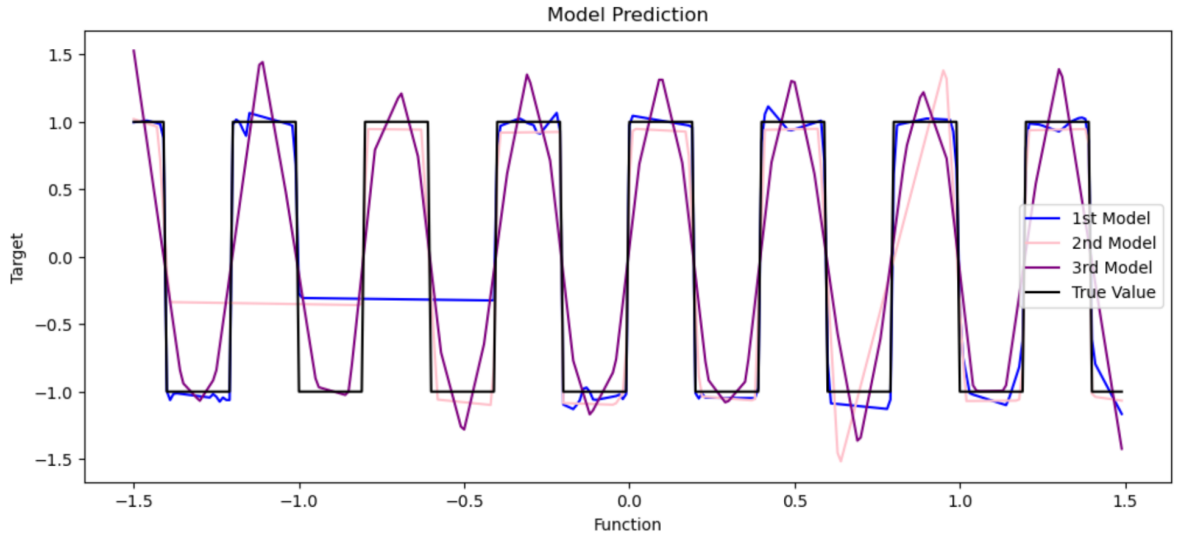
Function 2: sgn(sin(5*pi*x) / 5*pi*x)
Below is the function plot for the same function:



Simulating of the function:
All models will converge once the maximum number of training cycles has been used or the learning rate starts to dramatically slow down. The loss for each of these models is depicted in the below graph.



The graph below shows the comparison between the three models' forecasts and the ground truth.



Result:

Because the function being modeled is challenging for all three models, their number of training cycles until convergence is reached. Model 1 is the most effective model because it outperforms Model 2 marginally and has the lowest loss. Model 3 had a higher loss value, however, and was unable to converge within the necessary number of training cycles. This supports the notion that neural networks with more layers typically converge more quickly and learn more effectively.

HW 1.1 Training on Actual Sets

The MNIST dataset was used to train the models
described below, with a training set size of 60,000
and a test set size of 10,000.

Model 1: A LeNet convolutional neural network with the following cellular architecture

Maximum pooling followed by ReLU activation in the second convolution layer
Maximum pooling followed by ReLU activation in the second convolution layer
ReLU activation in the 2D Dense Layer
ReLU activation in the 2D Dense Layer

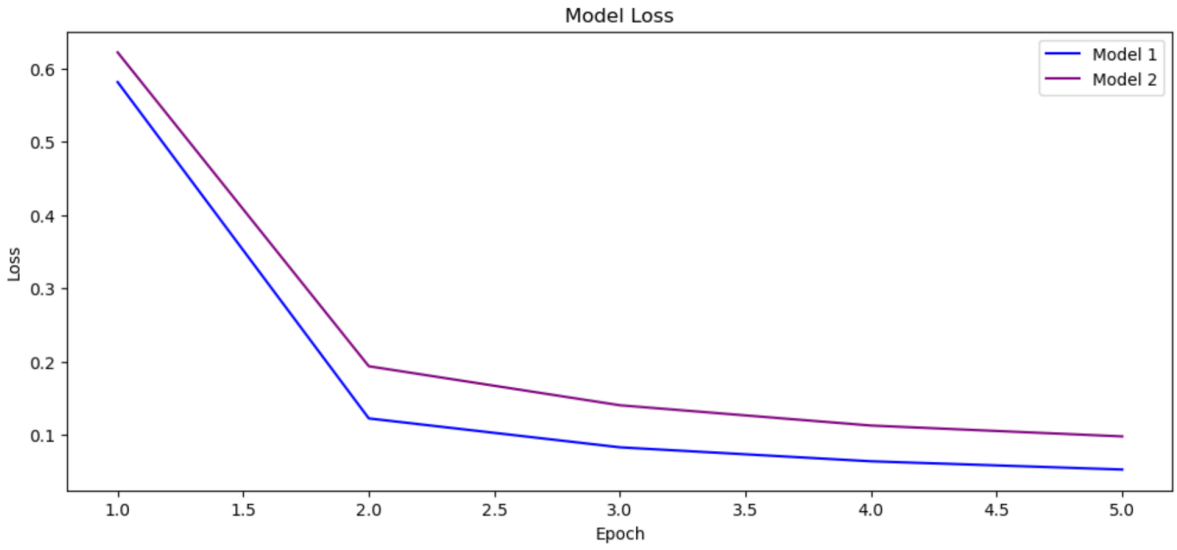
Model 2: An individual convolutional neural network with the following cellular architecture

ReLU activation in the 2D Convolution Layer
max pooling, followed by ReLU activation and Dropout, in the 2D Convolution Layer
max pooling, followed by ReLU activation and Dropout, in the 2D Convolution Layer
2D ReLU activation with Dropout in the Dense Layer
2D Dense Layer: activation of Log softmax (Output)

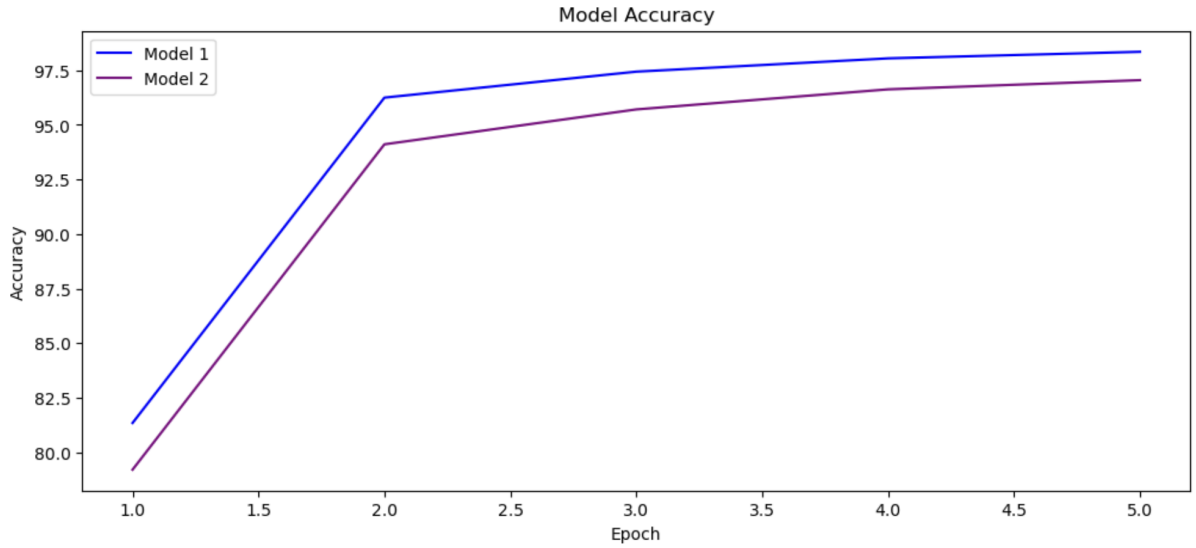
Hyperparameters:

0.5 Momentum with a 0.01 learning rate
Stochastic Gradient Descent Optimizer batch size: 64 epochs: 10
Loss Cross Entropy: Function

The training loss for Models 1 and 2 is shown in the graph below.



The training accuracy for Models 1 and 2 is shown below.

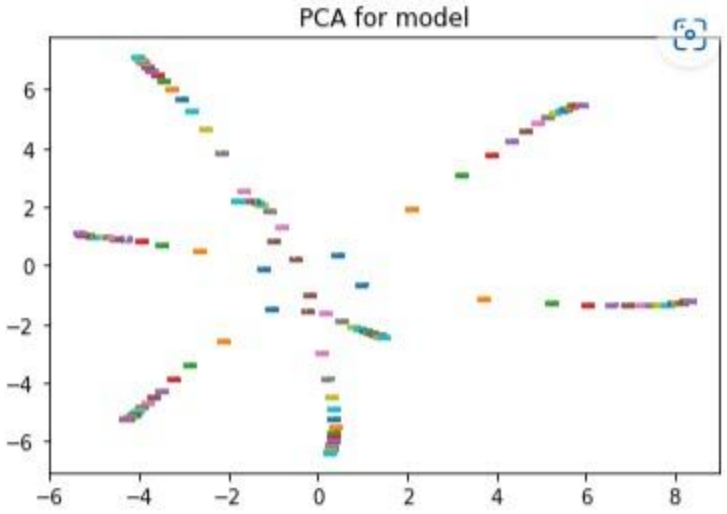


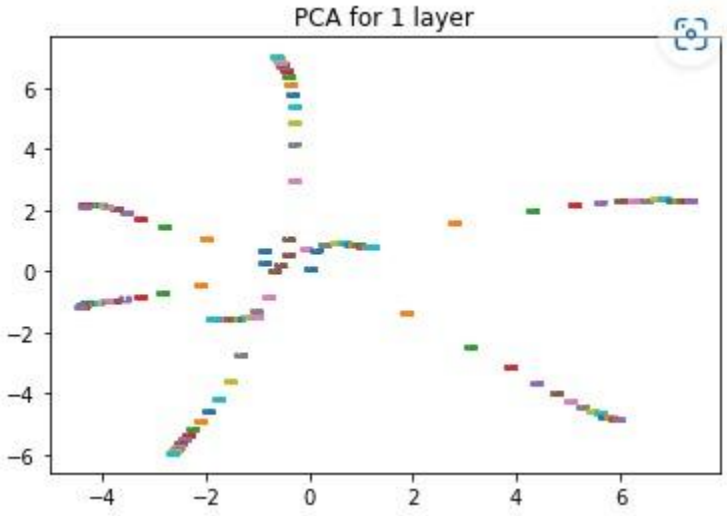
Result:
LeNet structure-modified Model 1 performs better than Model 2 and has a reduced loss. Comparing it to the specially created CNN model shows that it performs better. The accuracy also reflects this, with Model 1 obtaining a better training accuracy than Model 2.

HW 1.2 Visualizing the optimizing process

Train eight times while collecting the weights every three epochs. By using PCA, reduce the weights' dimension to 2.

Every three epochs, weights were collected as the model was trained eight times. The weights were then condensed using PCA to 2 dimensions. 10,000 samples were used for testing after 10,000 samples were used for training on the MNIST dataset. The model has a cross entropy loss function and three dense layers. Adam is the optimizer employed, and the batch size is 1000, the learning rate is 0.0004, and the ReLU activation function.



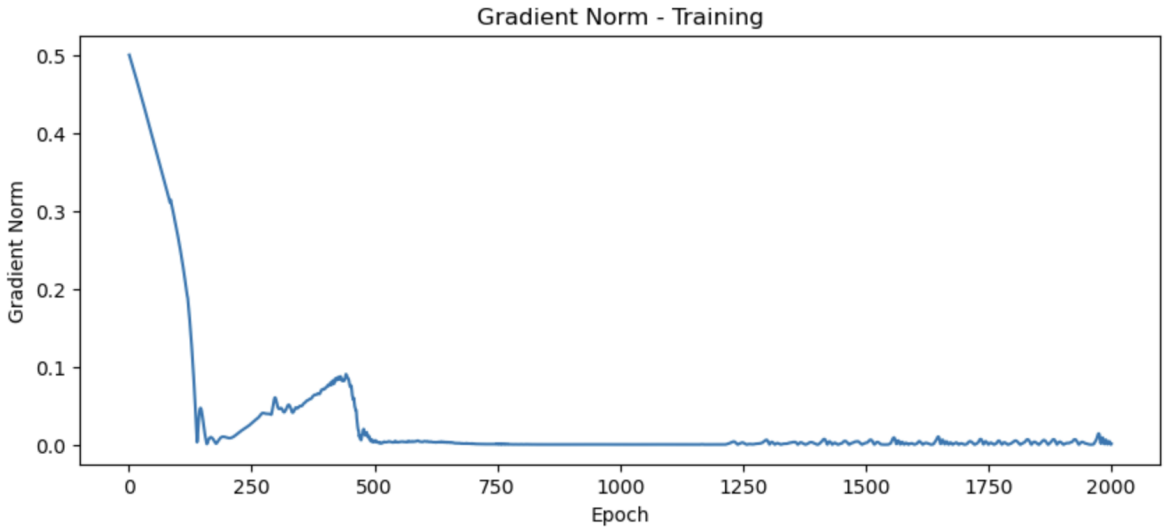


The graph above displays the outcomes of shrinking the dimensions of the weights acquired from the model trained eight times using a set of weights every three epochs. The dimensionality reduction was carried out using PCA, which reduced the 8240 parameters or weights to only two. The MNIST dataset, comprising 60,000 instances for training and 10,000 for testing, was used to train the model. With a cross-entropy loss function, Adam optimizer, learning rate of 0.0004, batch size of 1000, and ReLu activation function, it has three dense layers.

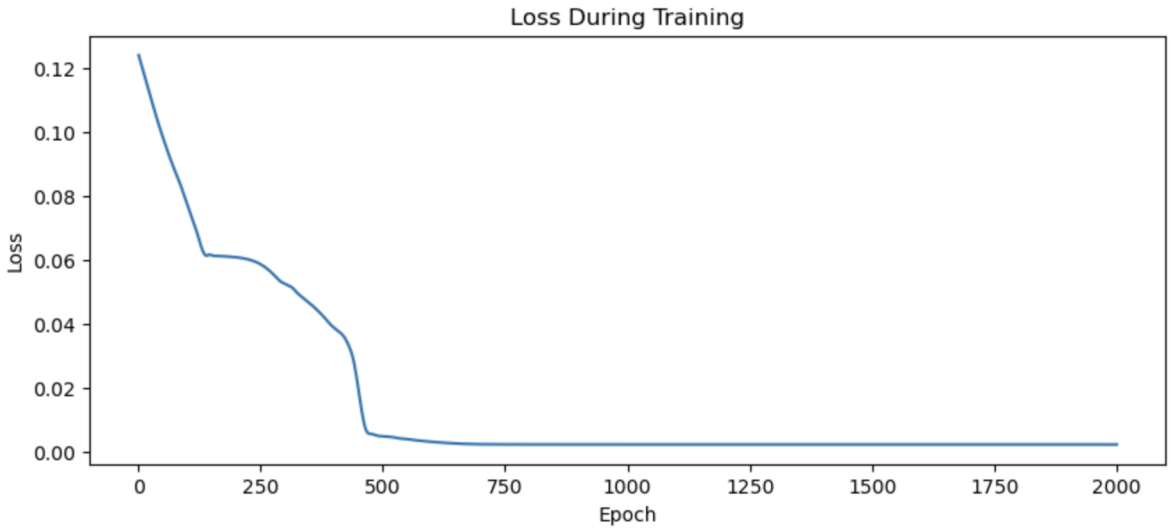
HW 1-2 Observe gradient norm during training

The gradient norm and the loss have both been determined using the function sin(5*pi*x) / 5*pi*x. Since the input to the model is already of a tiny size, I trained it on epochs rather than iterations.

The graph displays the gradient norm with time.



The graph for loss over time is shown.



Result

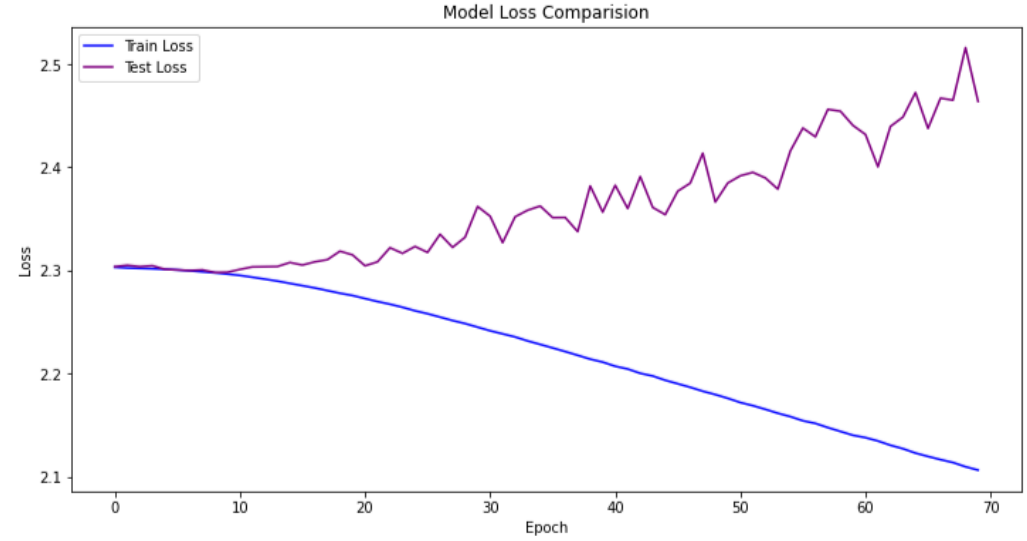
The graph for the gradient norm throughout epochs is shown below. The model has been refined and trained. After 100 epochs, the gradient gradually increases, a phenomenon that is also seen with the loss in the other graph, which initially plateaus and then declines more slowly before plateauing at the end after roughly 500 epochs.

HW 1.3 Can random labels fit to the network?

The MNIST dataset was used to train the model displayed above, with a training set size of 60,000 and a test group size of 10,000.

LeNet is a CNN (Convolutional Neural Network) model that has two 2D dense layers and two 2D convolutional layers.

ReLu is the chosen activation function, and Adam is the chosen optimizer. With a training batch size of 100 and a testing batch size of 100, the learning rate is set at 0.0001.
Cross-entropy loss was used as the loss function during the model's 100 training iterations.



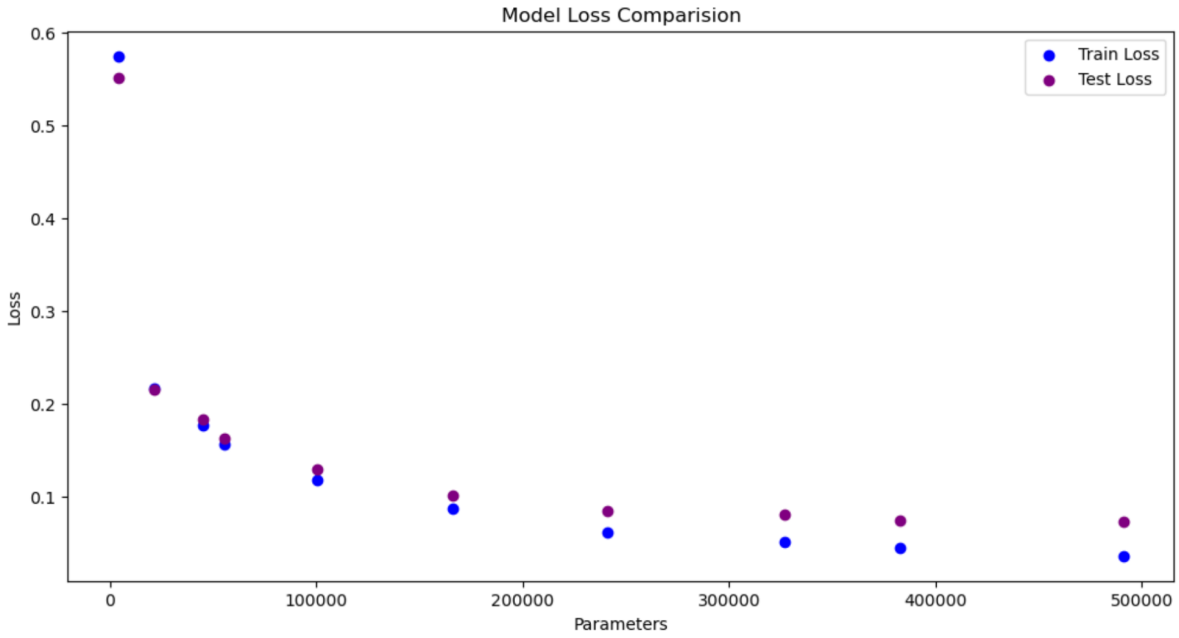
The model was trained using labels that were given to it at random, and as training goes on, it tries to memorize the labels, which slows down learning. The graph shows that as the number of epochs increases, the difference between the training loss and test loss widens, which points to overfitting.

HW 1.3 Generalization vs Number of Parameters

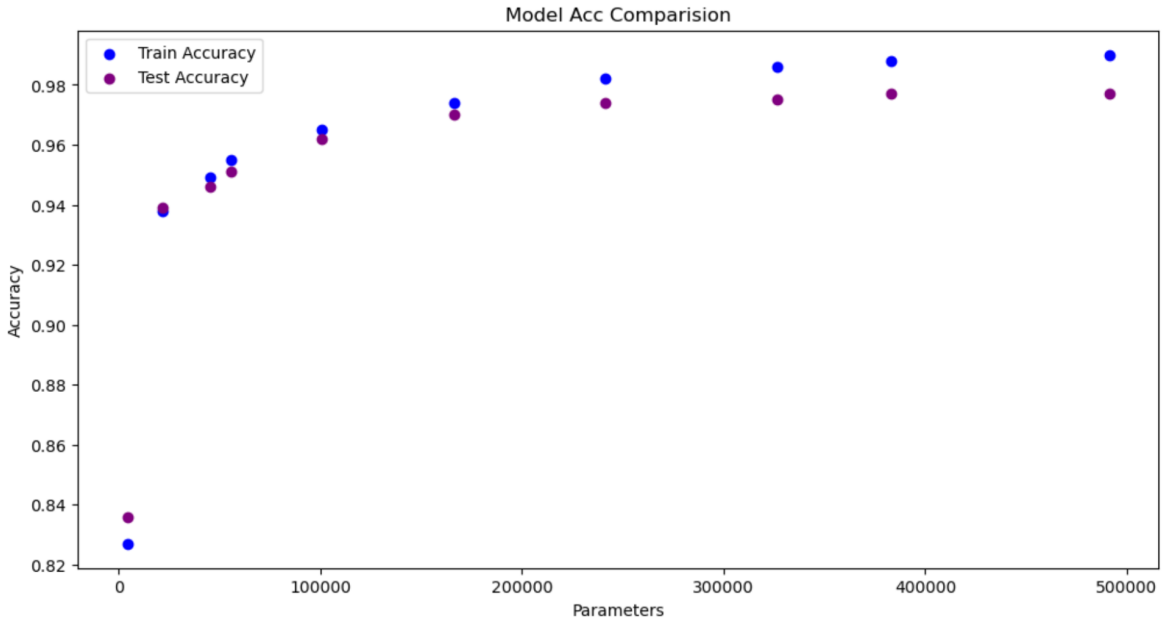
Hyperparameters:
An Adam optimizer and a learning rate of 1e-3 were used to train the model. Cross Entropy Loss was the chosen loss function, and the batch size was 50. ReLU served as the activation function. The model has three dense layers and was made up of 60,000 examples from the MNIST dataset that were used for training and 10,000 examples for testing.

Three dense layers make up the model structure, and the inputs and outputs are generally doubled for each layer. The model becomes more complex and challenging to train as a result of the increase in training parameters.

The comparison of losses across several models is shown in a graph.



Graph for Accuracy comparison for the various models



As observed from the graphs above, the difference between the train and test loss/accuracy grows as the number of factors increases. Much earlier than the training loss or accuracy, the test loss begins to plateau.

As there are more parameters for the model to train on, overfitting is the cause of this. Even while this improves test accuracy and reduces training loss, we still need to work to narrow the gap between test and training accuracy and loss in order to prevent overfitting. I was unable to add more parameters due to computational resource constraints, but the trend is evident from the graphs above.

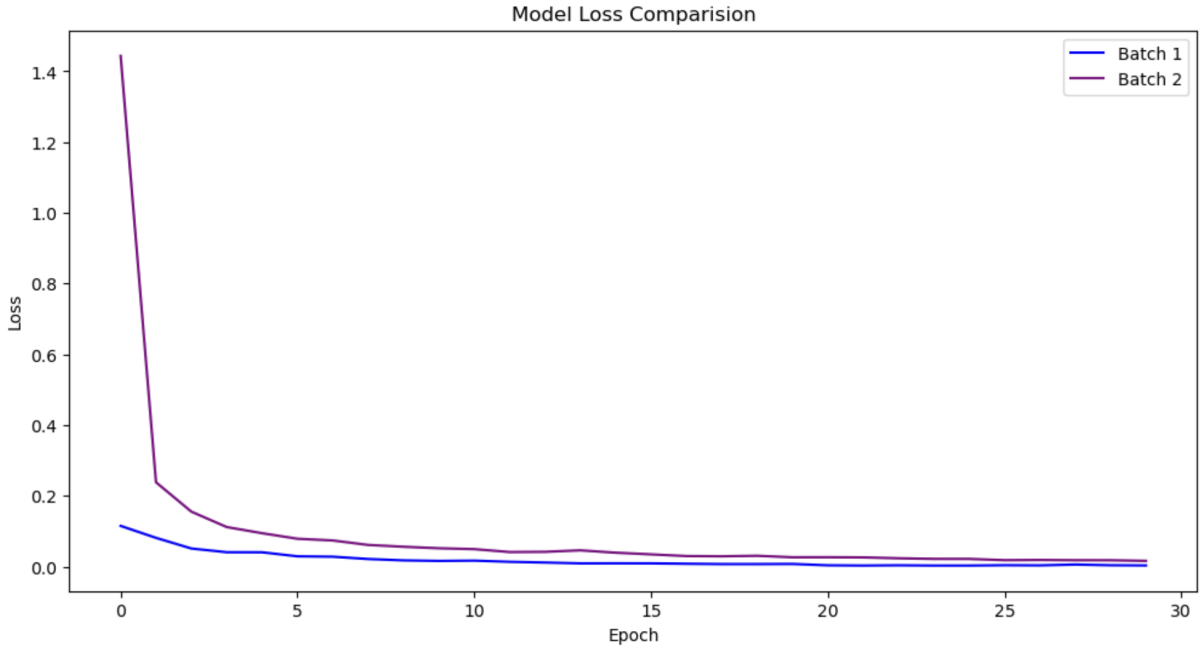
HW 1.3 Flatness vs Generalization:

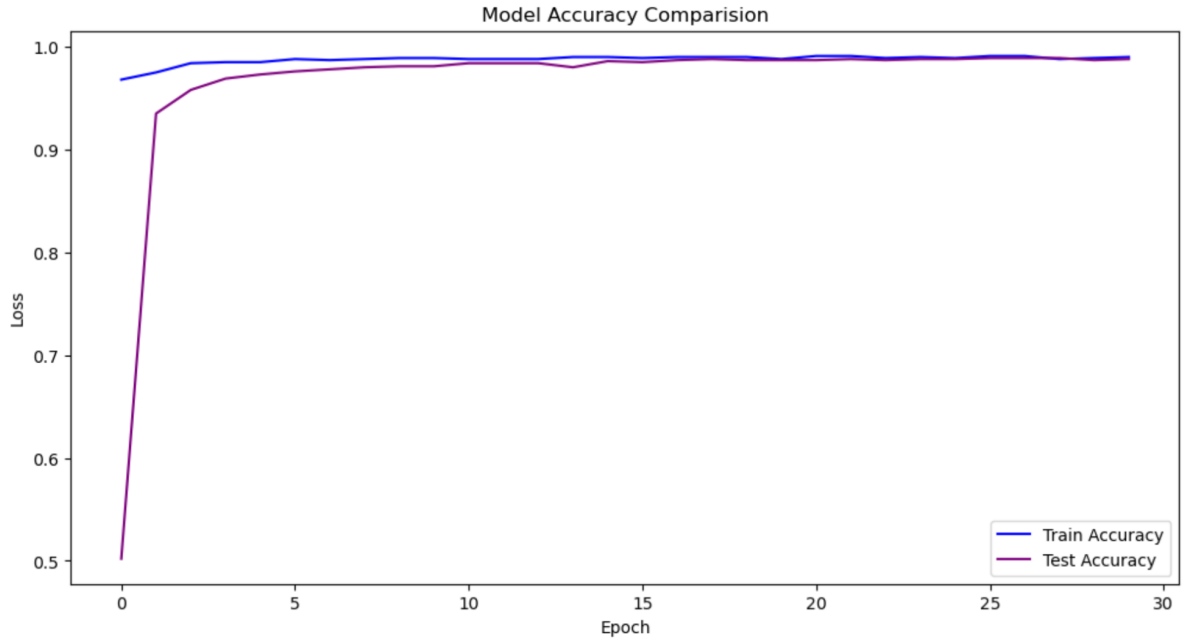
Part 1:

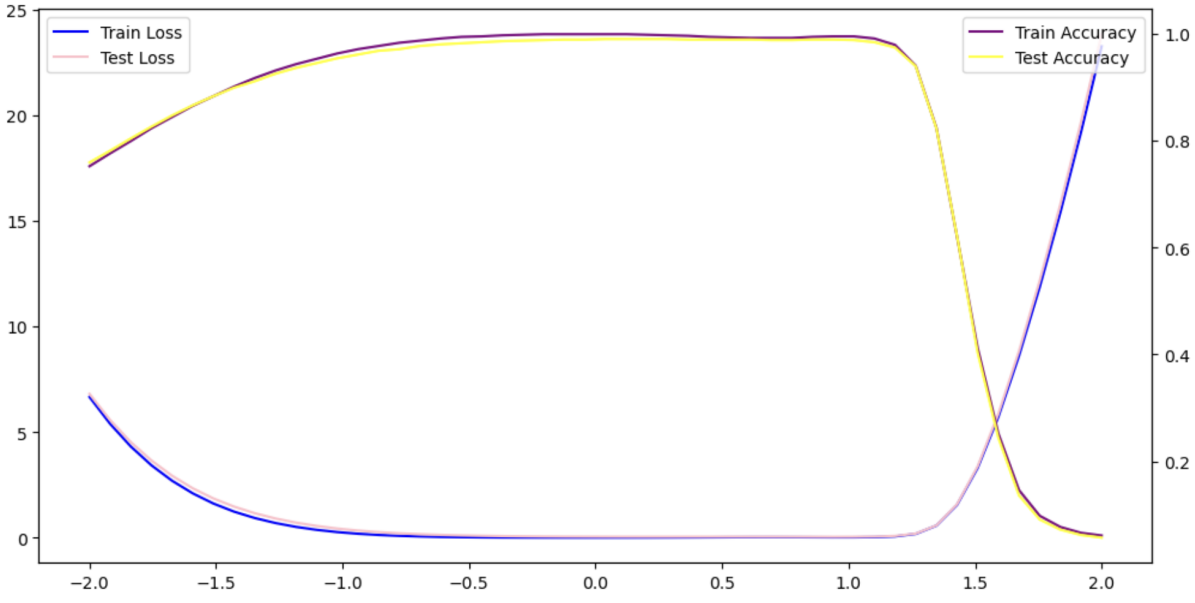
The MNIST dataset, comprising 60,000 training samples and 10,000 testing samples, was used to train the model shown below.

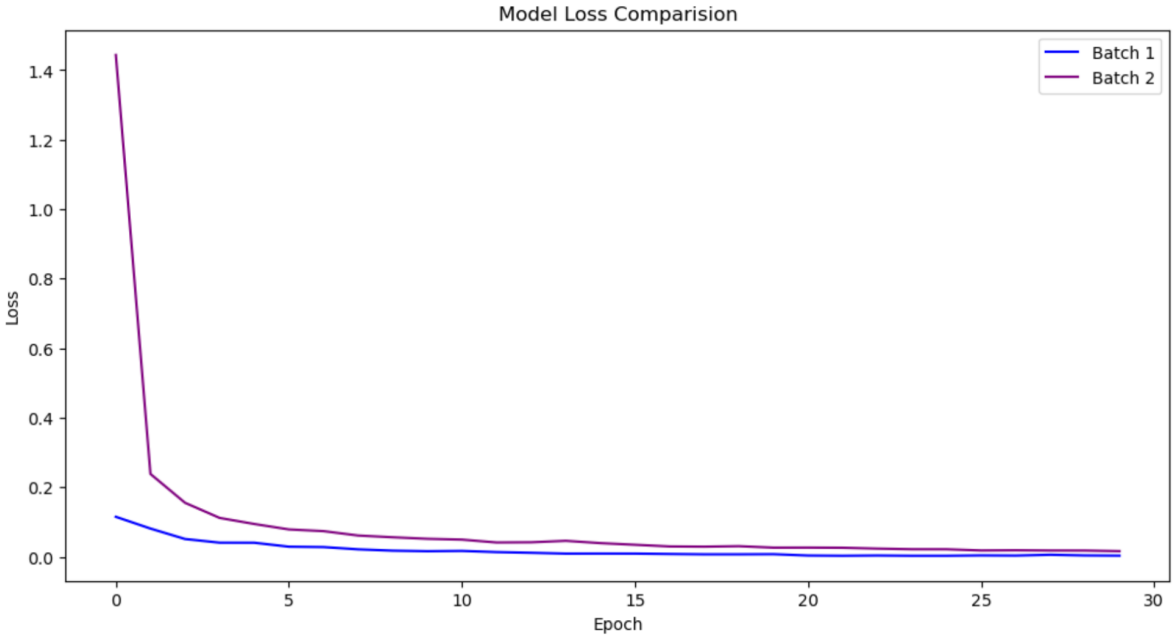
It has two convolution layers, three dense layers, and the Cross Entropy Loss loss function. Stochastic Gradient Descent (SGD), with learning rates of 0.001 and 0.01 is the optimizer of choice. The activation function utilized in the model's training was ReLu, and batch sizes of 100 and 500 were employed. The process is described in this overview.

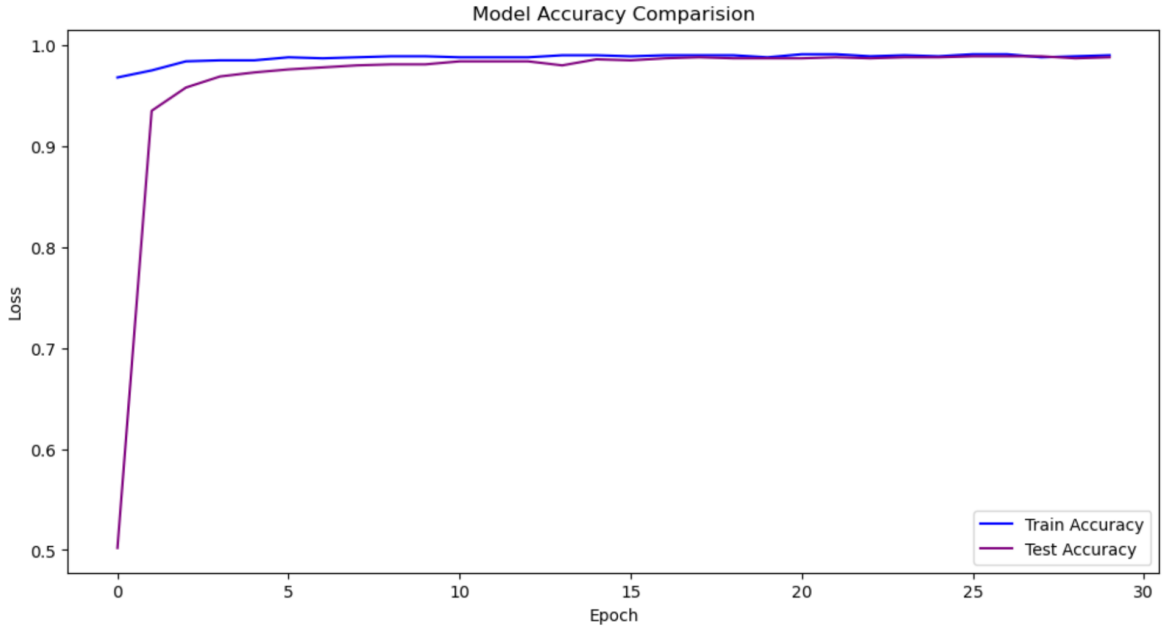
After the models have been trained using various batch sizes, their weights are calculated. The formula "(1-alpha)* batch1 param + alpha* batch2 param" is then used to compute the interpolation ratio. Using the new weight values obtained from the interpolation ratio, 50 new models are produced. The loss and accuracy of the new model, as well as the loss and accuracy of the two models with various batch sizes, are noted and plotted on a single graph. The adopted learning rate is 1e-2.

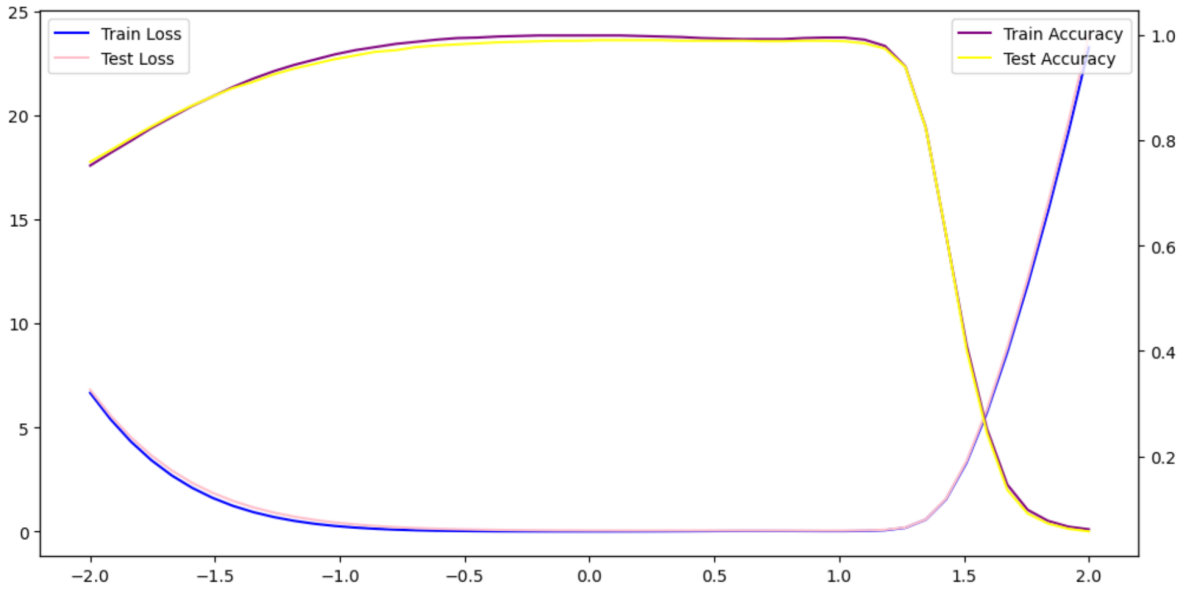



Model Accuracy Comparision
Loss
Epoch
Train Accuracy
Test Accuracy


Train Loss
Test Loss
Train Accuracy
Test Accuracy


Model Loss Comparision
Batch 1
Batch 2
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
5
10
15
20
25
30


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



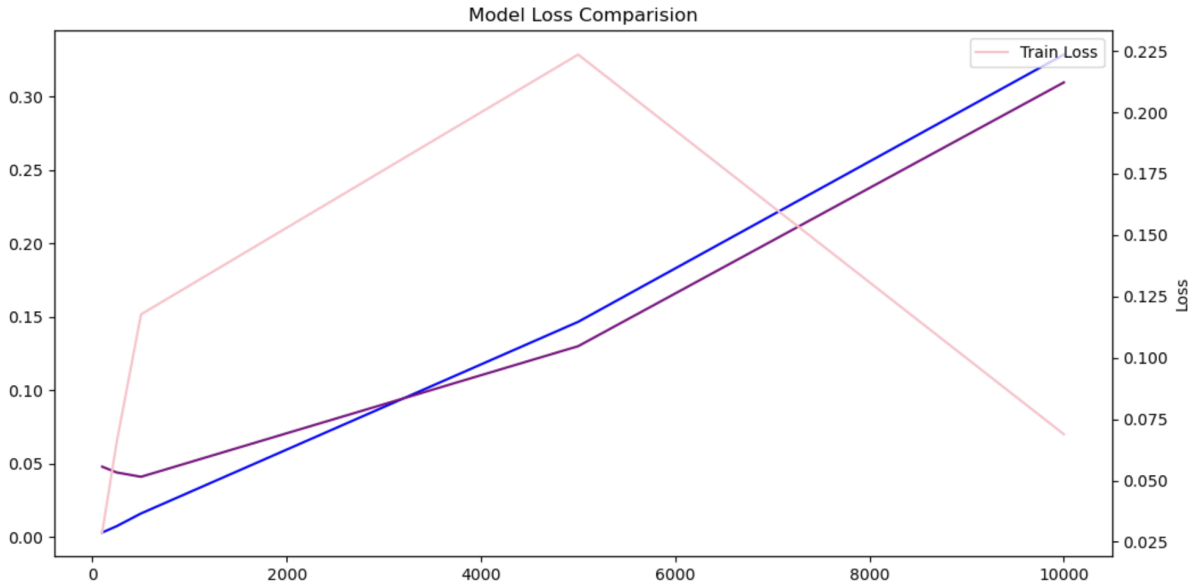
Learning Rate 1e-3

The result, for both models with distinct learning rates of roughly 1.5, reveals a reduction in the accuracy gap between the test and train datasets. According to the graph, accuracy starts to rise sharply at an alpha value of 1.5, which is obtained by multiplying alpha by the batch 1 and batch 2 parameters, or "(1-alpha) * batch 1 parameters + alpha * batch 2 parameters." This shows that although while all machine learning algorithms have a tendency to extrapolate between data points, if there are more parameters than there are actual data, they will simply memorize the data.

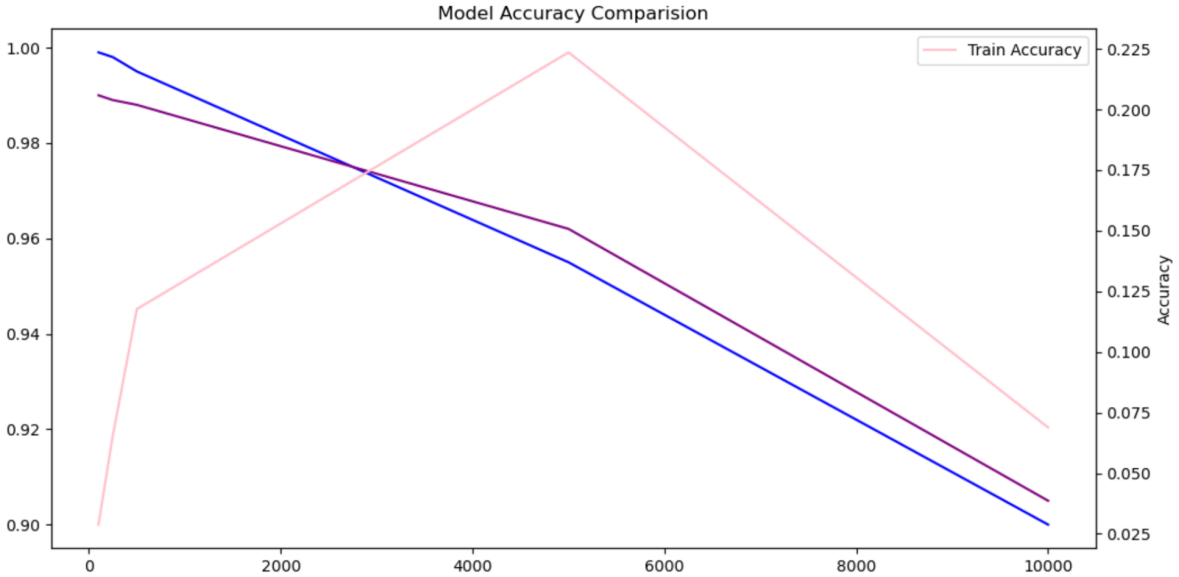
Flatness vs Generalization – Part 2

The MNIST dataset, which contains 10,000 samples for testing and 60,000 samples for training, is used to train the proposed model.

It employs cross-entropy loss as its loss function, contains three dense layers, and two convolution layers. Stochastic Gradient Descent (SGD), with a learning rate of 1e-3 and a batch size of 50, is the optimizer employed. Rectified Linear Unit is the activation function that is utilized (ReLU).



Note: The blue line in the accompanying graph is mistakenly identified as sensitivity, the red line as model accuracy, and the green line as loss in the legend. The loss is displayed on the Y-axis of the top graph, while batch size is displayed on the X-axis. The accuracy axis is on the bottom graph.



The plots show that the sensitivity diminishes as batch size rises. When the batch size is around 4000, the sensitivity is at its peak. Thereafter, it gradually declines along with accuracy and loss value. This demonstrates how the network becomes less responsive when batch sizes rise above 5000.