CPSC 8430: Deep Learning

Assignment 1

Saurabh Sharma
02/08/2024

Saurabh Sharma

1) Simulate a function

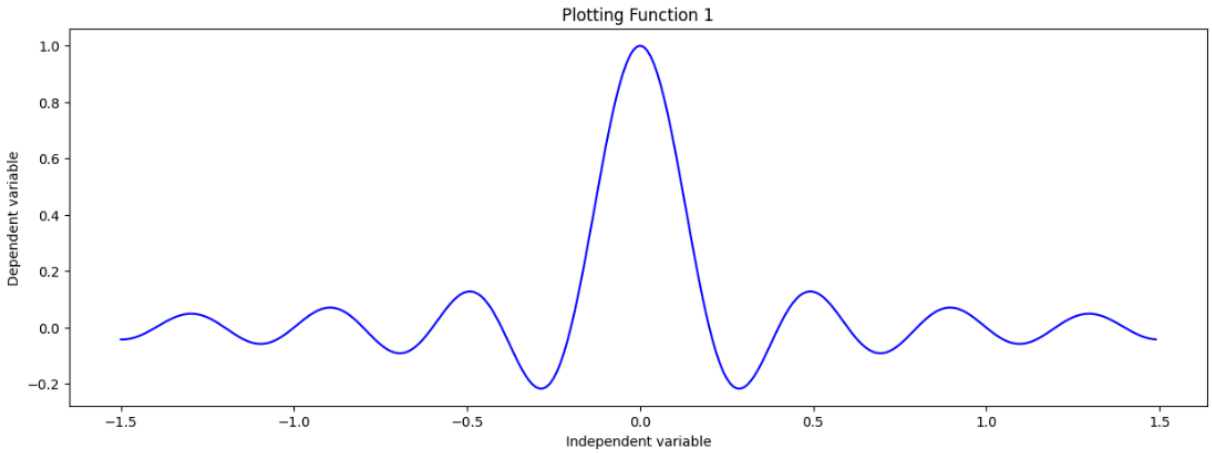
Models: I have built three models that correspond to the ones mentioned in the specifications. By including a penalty term in the neural network's cost function, which has the effect of reducing the weights during backpropagation, a weight decay parameter has been added to improve the regularisation of the models.

Model 1 is a deep neural network designed with complexity and depth in mind, featuring 7 dense (fully connected) layers comprising a total of 571 parameters. This model utilizes the Mean Squared Error (MSE) loss function and the RMSProp optimization algorithm, with a learning rate of 1e-3, aiming to efficiently minimize errors during training. It employs LeakyReLU as its activation function, providing a slight gradient when the unit is not active to prevent dead neurons and facilitate better learning. Additionally, a weight decay of 1e-4 is applied as a regularization technique to reduce overfitting by penalizing large weights. This architecture suggests an attempt to capture complex patterns and relationships in the data, potentially making it suitable for tasks requiring intricate model understanding but also necessitating careful management of overfitting risks.

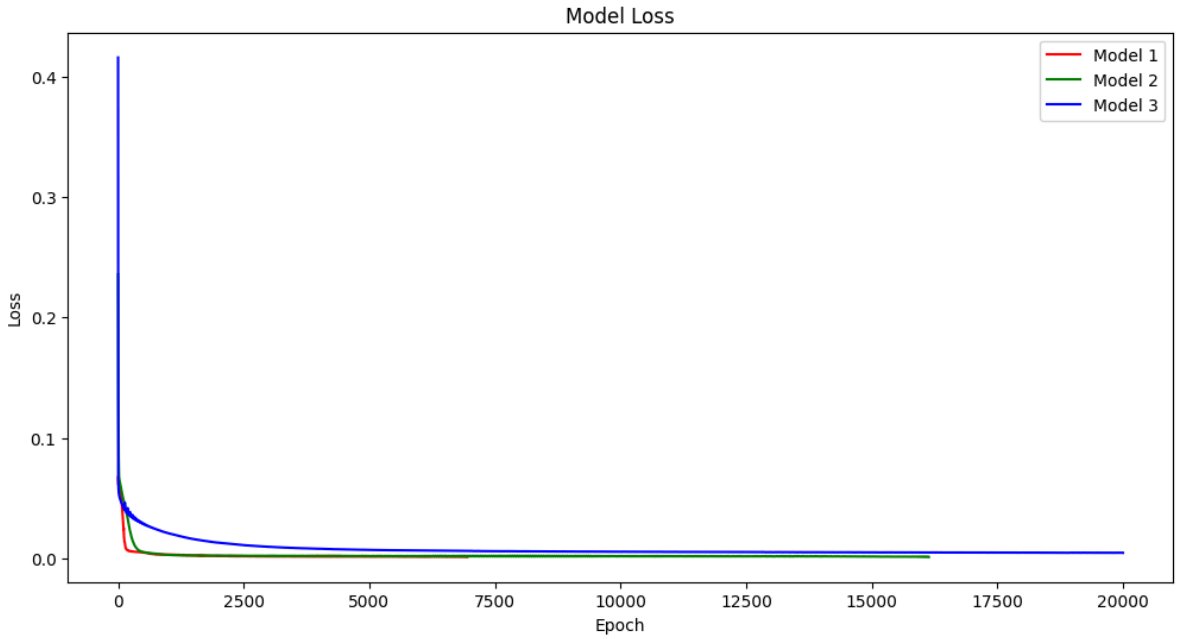
Model 2 simplifies the architecture to 4 dense layers while maintaining a nearly identical total number of parameters (572), indicating a different distribution of units across its layers compared to Model 1. Like Model 1, it also uses the MSE loss function, RMSProp optimizer, a learning rate of 1e-3, LeakyReLU activation function, and a weight decay of 1e-4. This setup suggests a model that seeks a balance between the capacity to learn non-linearities and the need for generalization, potentially making it more adaptable to a variety of tasks without the heightened risk of overfitting that comes with deeper networks. Its structure might offer an effective blend of complexity and efficiency, suitable for problems where a moderately complex model architecture is needed.

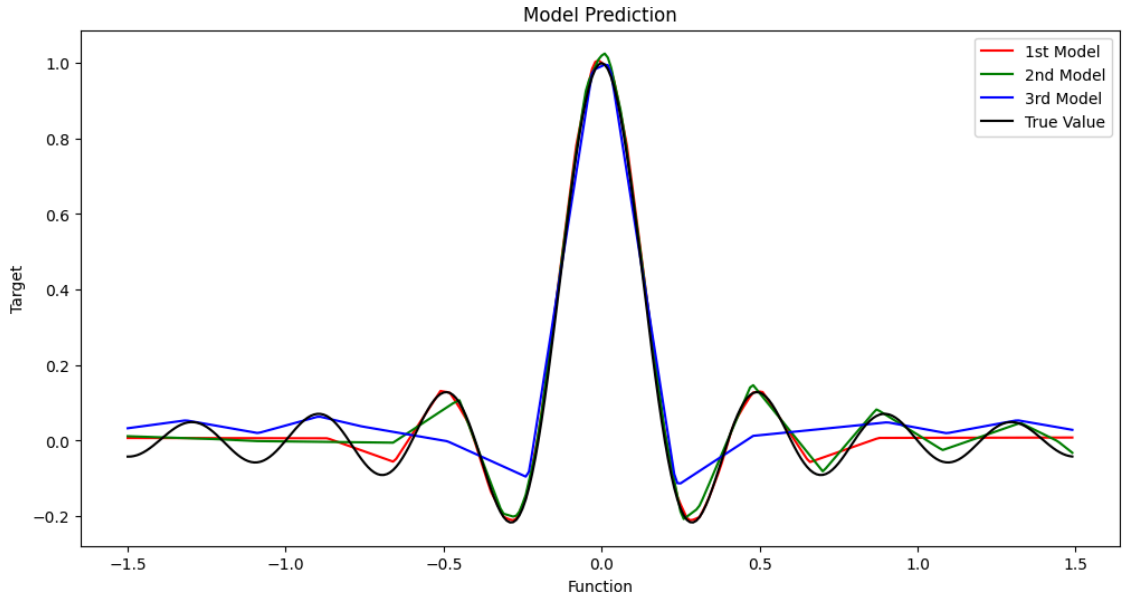
Model 3 significantly reduces the network's complexity to just 1 dense layer, yet it contains 571 parameters, almost the same as Model 1. This minimalistic approach still utilizes the same MSE loss function, RMSProp optimizer, learning rate of 1e-3, LeakyReLU activation, and weight decay of 1e-4 as the other two models. The choice of a single-layer model suggests a focus on tasks where a simpler model can capture the underlying patterns without the need for deep, hierarchical feature extraction. This model might excel in scenarios where the relationship between inputs and outputs is relatively straightforward or when the goal is to prevent overfitting while maintaining a decent level of model interpretability. It's designed for efficiency and speed, possibly at the expense of capturing complex relationships present in more intricate datasets.

Function 1: sin(5*pi*x) / 5*pi*x
Below is the function plot for the same function:



All models either reach the maximum number of epochs when they learn very slowly or they converge before. Below graph shows how quickly the loss decreases with increasing epochs in our three different models.

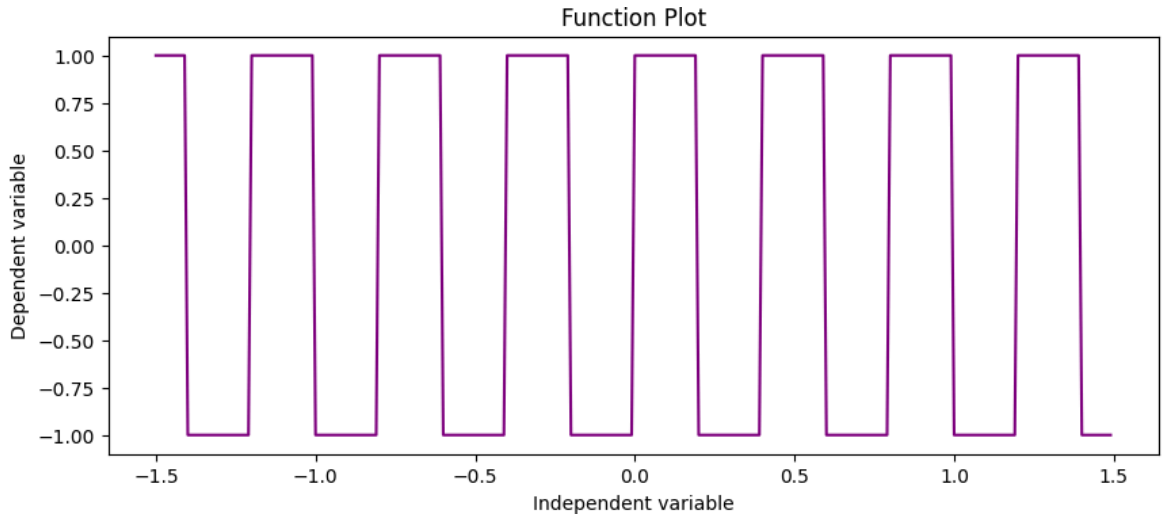




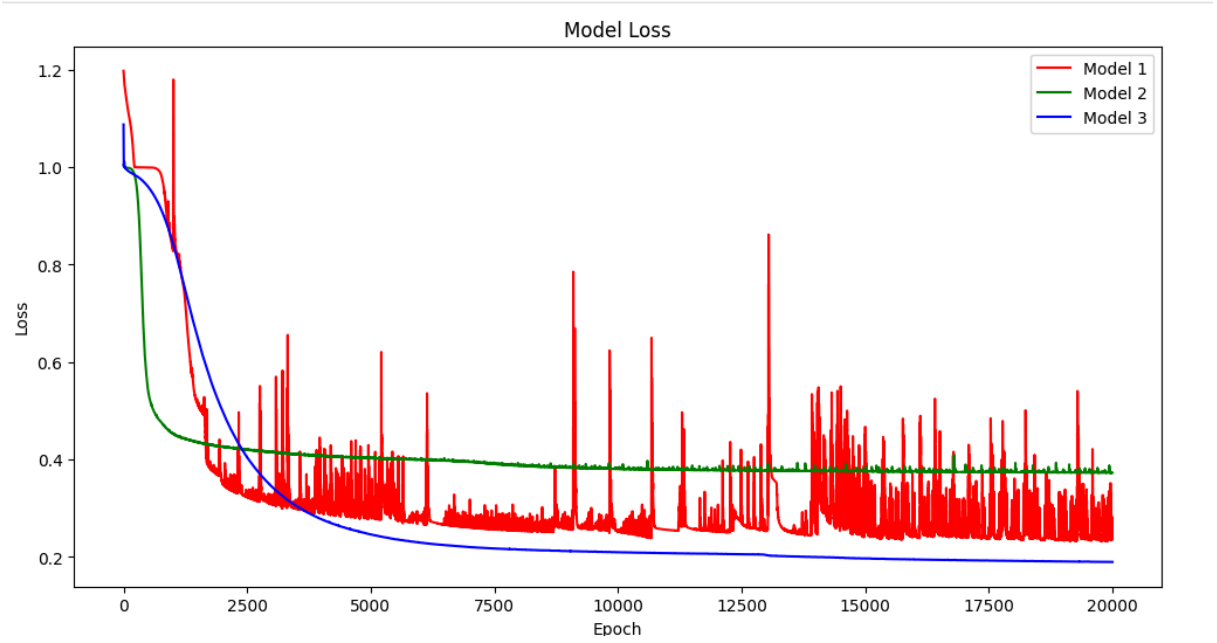
Result :
We can clearly see that first and second model converge very quickly while third model takes a lot of time to converge. This is because first and second model have a higher number of hidden layers and thus they are able to generalise (or learn in our terminology) much better while third model is very shallow and is unable to learn.

Function 2 : sgn(sin(5*pi*x) / 5*pi*x)

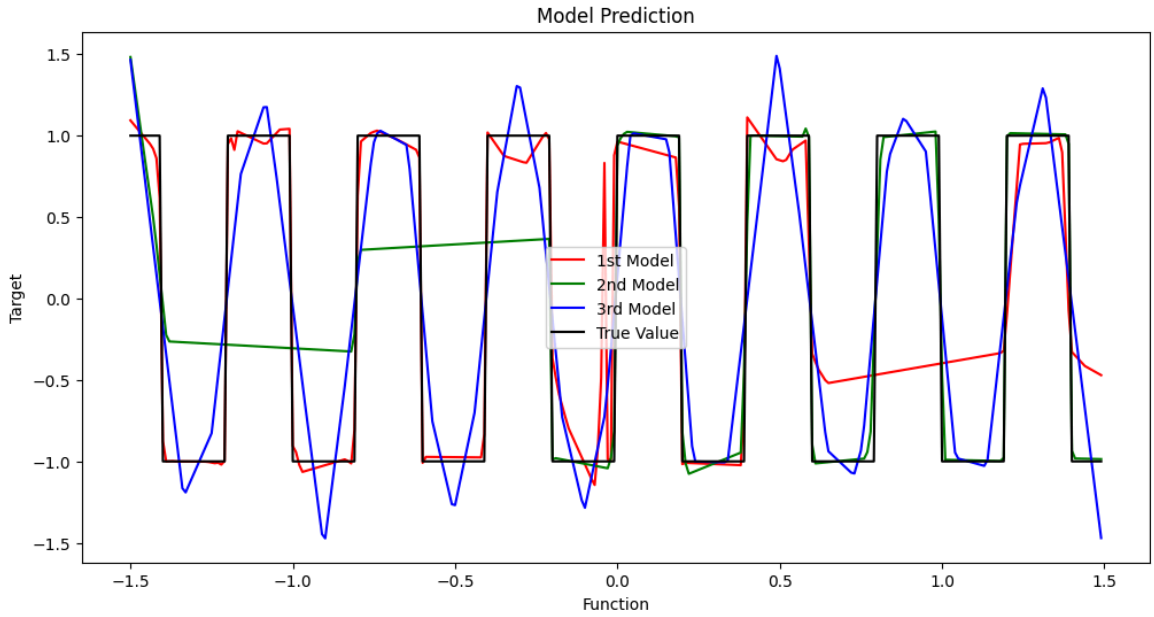
Below is the plot of given function 2 -



All models either reach the maximum number of epochs when they learn very slowly or they converge before. Below graph shows how quickly the loss decreases with increasing epochs in our three different models.



Overlapping the model predictions with the actual simulation of the function we can visualise the performance of our models much better.



Here we can see that all of our models have a hard time predicting the correct values. This is because the function itself is very complex and thus hard to generalise. But from the three models we see that model

3 learns relatively slowly but ends up generalising the best. Model 1 also performs good while model 2 has the worst performance. This again shows that the neural network with more hidden layers learns/ performs better.

1) Train on Actual dataset/ task

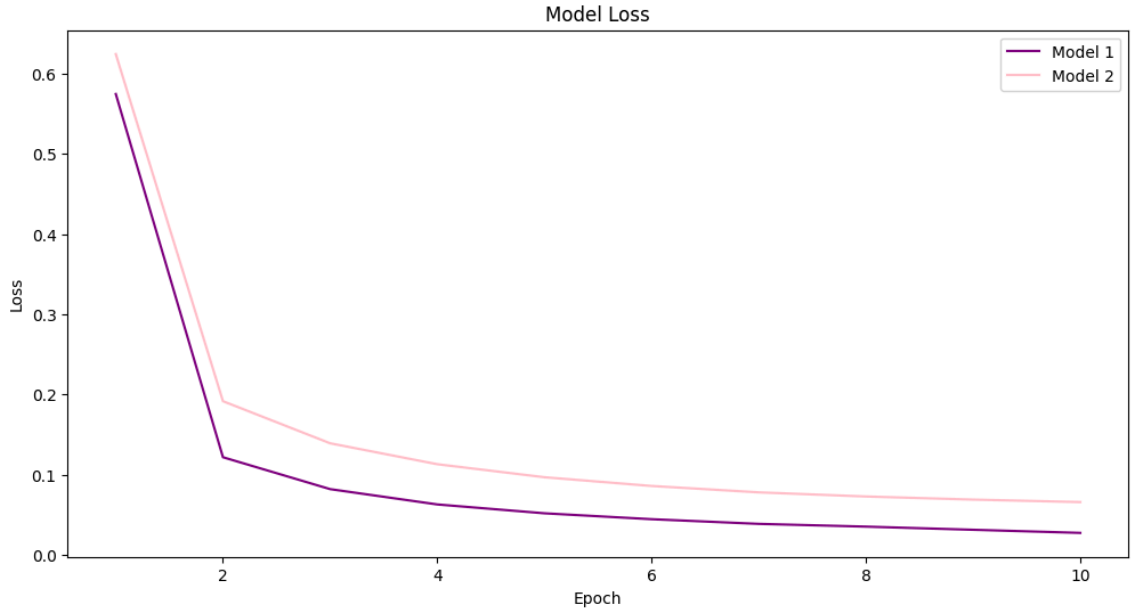
We have chosen two convolutional neural networks and we will train these on MNIST dataset which is widely available.

Model 1, structured around the LeNet-inspired CNN architecture, features a concise yet effective layer sequence. It starts with two 2D convolution layers, each followed by 2D max pooling and ReLU activation to enhance feature extraction through spatial hierarchies. This is succeeded by two dense layers, utilizing ReLU for non-linear transformation, aimed at processing the extracted features for complex decision-making tasks. This architecture efficiently combines convolutional layers for spatial feature refinement with dense layers for high-level reasoning, making it highly suitable for sophisticated image recognition tasks.

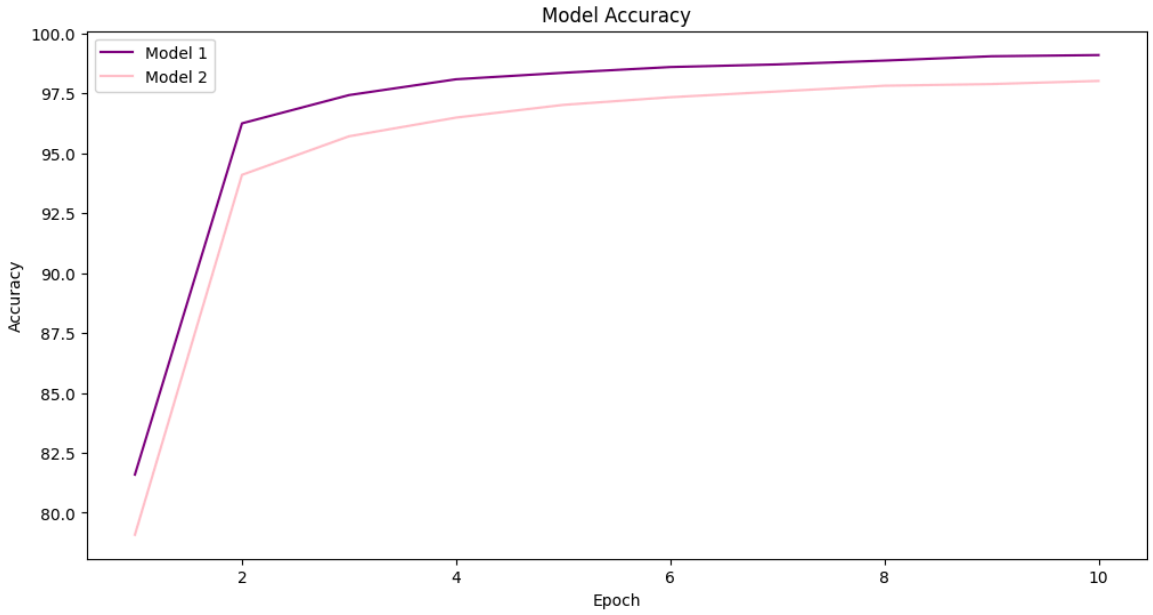
Model 2 is a custom CNN architecture that begins with a single 2D convolution layer using ReLU activation for initial feature extraction. It then progresses through two additional 2D convolution layers, each followed by 2D max pooling, ReLU activation, and a dropout layer to prevent overfitting by randomly omitting some of the layer's activations during training. This approach not only refines the spatial features but also enhances the model's generalization capabilities. The network concludes with two dense layers; the first employs ReLU activation and dropout for further feature processing and regularization, while the final layer utilizes a Log_softmax activation for output, providing a probabilistic distribution suitable for classification tasks.

Following standard hyperparameters were used -
• Learning Rate: 0.01
• Momentum: 0.5
• Optimizer: Stochastic Gradient Descent
• batch_size = 64
• epochs = 10
• Loss: Cross Entropy

Following graph shows the training loss for both of our models.



Below graph shows the model accuracy of both of our models.



Result -
Model 1, with its architecture rooted in the optimized LeNet CNN framework, exhibits superior performance compared to Model 2, demonstrating a lower loss and higher training accuracy over a series of epochs. This optimized CNN design significantly surpasses the performance of the custom-built CNN model, indicating a consistent trend in both loss reduction and accuracy improvement. The effectiveness

of Model 1's structure not only in minimizing loss but also in enhancing accuracy suggests its robustness and efficiency in learning from the dataset.

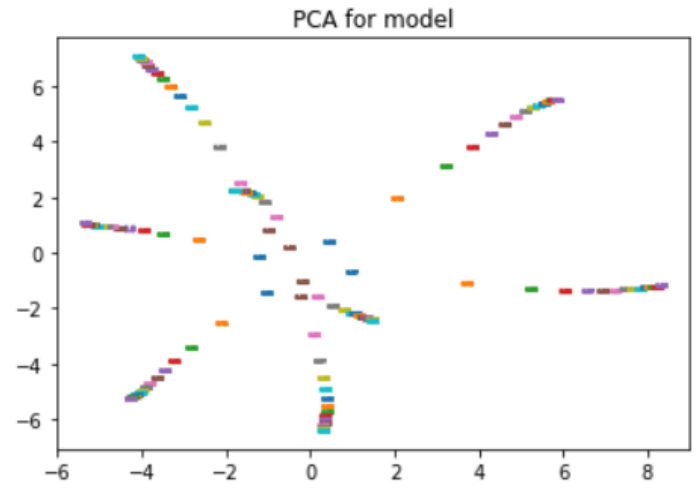
2) Visualize the optimization process

We will use a simple neural network and Collect the weights at intervals of every 3 epochs during the training process, which is to be repeated 8 times. Following the collection, reduce the dimensionality of the weights to 2 using Principal Component Analysis (PCA).

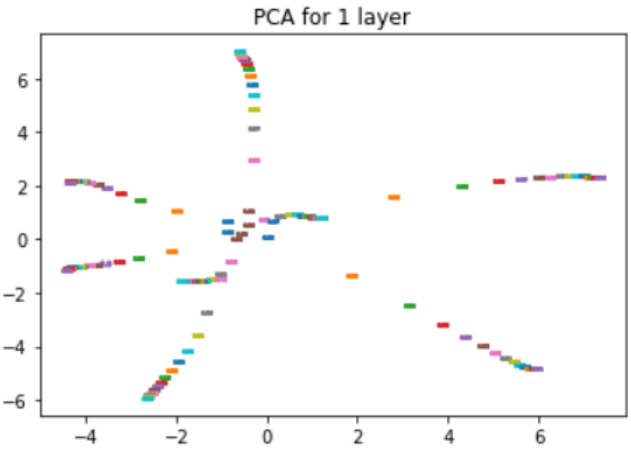
About the model,
The model consists of 3 dense layers and employs the Cross Entropy Loss function for evaluating model performance. It utilizes the Adam optimizer to adjust weights and biases, with a learning rate set at 0.0004. The training process is conducted with a batch size of 1000, and the ReLU activation function is applied within the dense layers to introduce non-linearity, facilitating the model's ability to learn complex patterns.
It'll be trained on MNIST dataset that we just used above and in similar proportion of testing and training.

Principle component analysis for the model is shown below,

```
plt.title("PCA for model")
```



PCA for the first layer is also given,

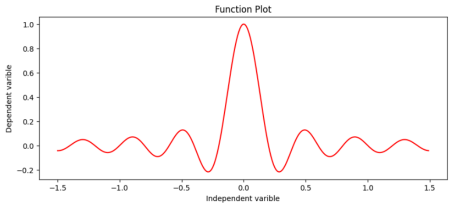


Following the PCA algorithm's application, the gathered weights are shown in the graph above. This indicates that the original weights' dimensions have decreased. After running PCA on the models—which were trained eight times across 45 epochs—we obtained the graph above. Originally, each model had 8240 parameters, or weights to train.

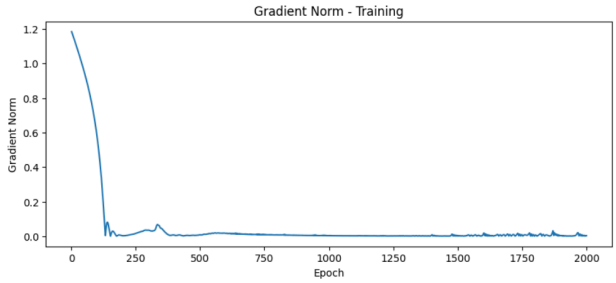
2) Observe gradient normalization during training

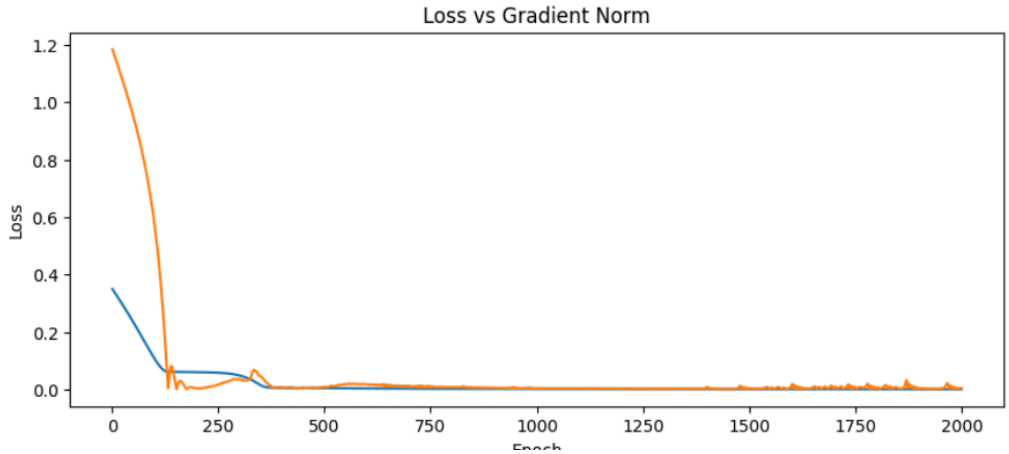
In the development of the model, the function sin(5*pi*x) / 5*pi*x has been used again to compute both the gradient norm and the loss metric. Given the smaller nature of the model, the training was done around epochs rather than iterations.

The function plot is,



The graph for gradient normalization and loss for our model with respect to epochs is shown below -





Result,
The model was trained and converged successfully. We can see that around 130 epochs the gradient increases and this perfectly coincides with the local increase in loss. Afterwards the gradient plateaus for a little bit and then decreases again before plateauing permanently around 350 epochs mark. Around that time, loss also reaches its minima and stays there.

3) Can network fit random variables

We are resuing the above Lenet inspired architecture. Model definition is given below,

Model 1, structured around the LeNet-inspired CNN architecture, features a concise yet effective layer sequence. It starts with two 2D convolution layers, each followed by 2D max pooling and ReLU activation to enhance feature extraction through spatial hierarchies. This is succeeded by two dense layers, utilizing ReLU for non-linear transformation, aimed at processing the extracted features for complex decision-making tasks. This architecture efficiently combines convolutional layers for spatial feature refinement with dense layers for high-level reasoning, making it highly suitable for sophisticated image recognition tasks.

Learning Rate = 0.0001
Optimizer = Adam
train_batch_size = 100
test_batch_size = 100
epochs = 100
loss function = CEL

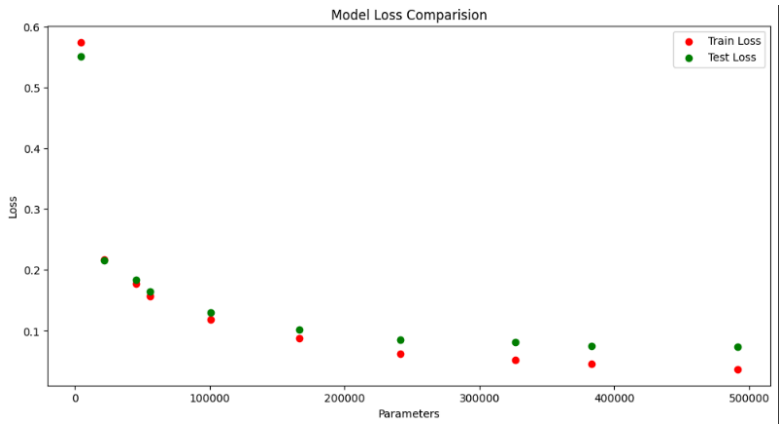
The model has undergone training with datasets characterized by randomly assigned labels. This randomness introduces a significant challenge in the learning process, resulting in a slower convergence rate as the model endeavors to adaptively memorize these arbitrary labels over successive epochs to minimize the training loss. Concurrently, there is an observable trend where the test loss progressively increases with each epoch, indicating a deterioration in the model's generalization capability. This phenomenon of diverging loss trajectories is corroborated by the data presented in the graph, which explicitly depicts the expanding disparity between the training and testing loss as the epoch count escalates, underscoring the model's increasing overfitting to the training data with random labels.

3) Number of parameters vs Generalization

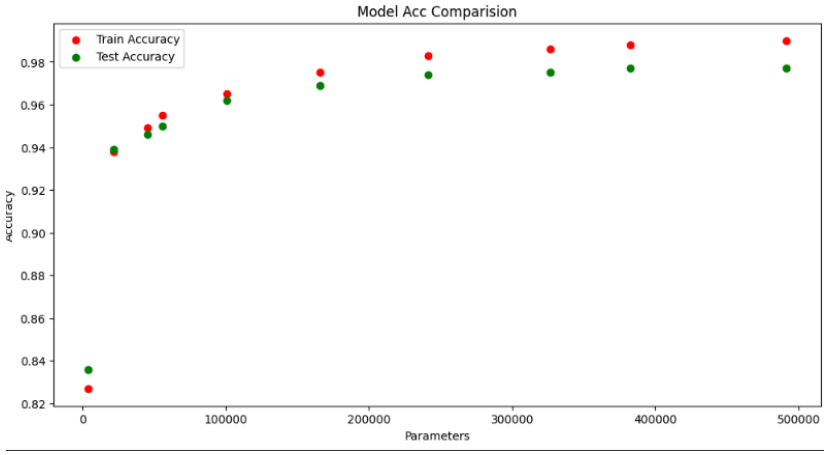
Below model is trained on MNIST dataset with 6:1 split of the dataset for training and testing respectively.

The architecture consists of three dense layers, utilizing Cross Entropy Loss as its objective function for optimization. It employs the Adam optimization algorithm to adjust and update the model's parameters, with a learning rate set at 0.001. Training batches are processed in sizes of 50, incorporating the ReLU (Rectified Linear Unit) activation function within its layers to introduce non-linearities, enabling the model to capture complex patterns in the data effectively.

Below graph makes it easy to visualize the loss comparison for various models.



And below graph helps us visualize the accuracy comparison between testing and training of various models.



Results inference -

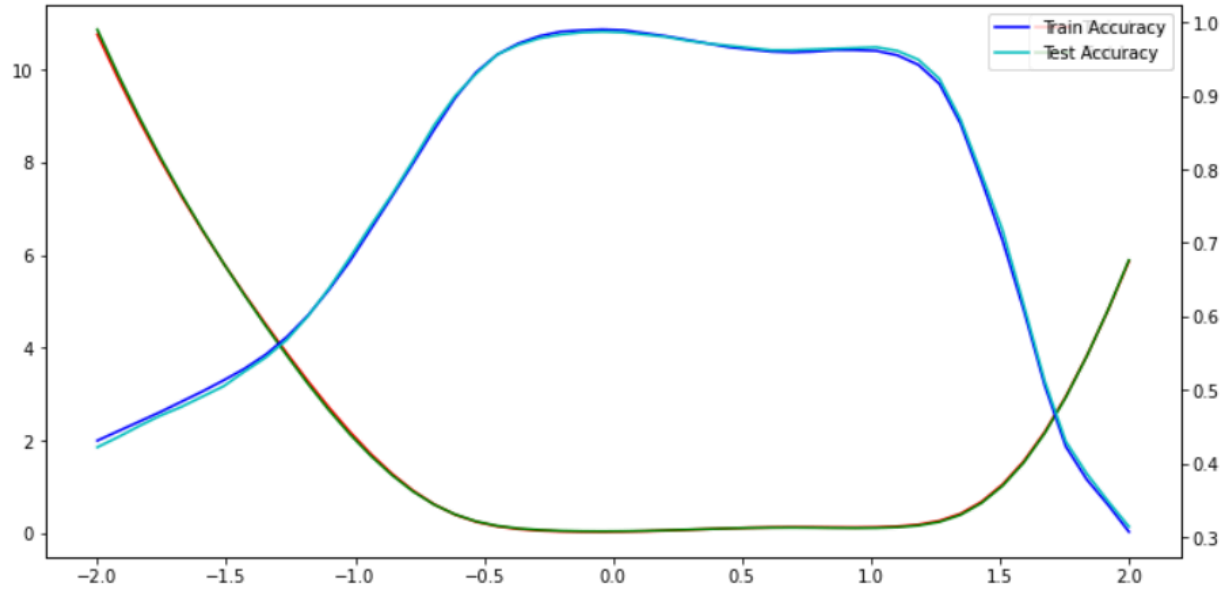
The observed trends in the graphs clearly demonstrate that as the number of parameters in the model increases, the discrepancy between training and testing loss/accuracy becomes more pronounced. Notably, the test loss begins to plateau significantly earlier than the training loss or accuracy. This phenomenon is indicative of overfitting, where the model, endowed with an excess of parameters, becomes overly specialized in the training data, enhancing its performance on these data points but failing to generalize effectively to new, unseen data. While this augmentation of parameters may lead to improved accuracy and diminished loss during training, it is crucial to strive for a minimization of the gap between training and testing performance metrics to mitigate the effects of overfitting. The constraints of computational resources prevented further exploration into increasing the model's parameters; however, the trend towards overfitting with an increase in parameters is evident from the existing data and graphs. Essentially, the model is memorizing all the training data without generalizing it (i.e. learning it). Hence a balance must be striked between complexity of task to be learned and number of parameters of model.

3) Flatness vs Generalization - Part 1

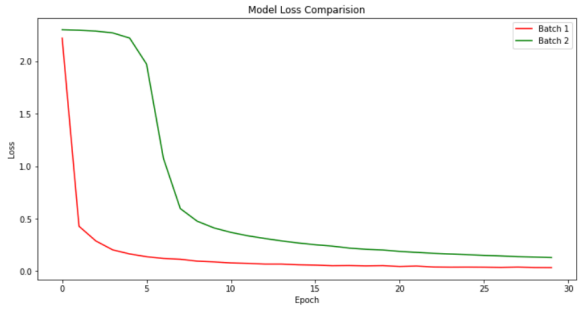
We used the same model as above, The model configuration consists of three dense layers and two convolution layers, designed to process data through a combination of feature extraction and nonlinear transformation. It utilizes Cross Entropy Loss as the objective function to guide the training process towards minimizing the discrepancy between predicted outcomes and actual labels. The Stochastic Gradient Descent (SGD) optimizer is employed to iteratively adjust the model's parameters, with experiments conducted using two distinct learning rates:1×10−3 and 1×10−2, to observe their impact on the model's learning dynamics. Training is performed in batches, with sizes set at 100 and 500, to balance between computational efficiency and gradient estimation accuracy. The ReLU (Rectified Linear Unit) activation function is applied

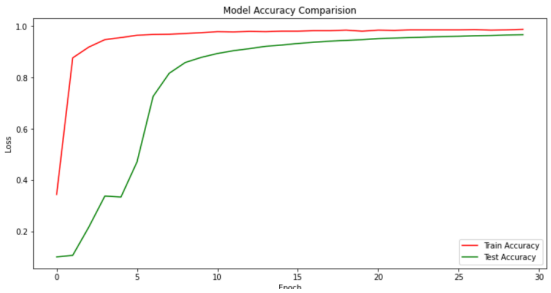
What we are doing,
To investigate the impact of batch sizes on model performance, we undertake a systematic training regime for models under varying batch sizes, subsequently collecting their parameter weights upon completion. Utilizing the formula "(1-alpha) * batch1_param + alpha * batch2_param", we calculate the interpolation ratio between the parameters of models trained with different batch sizes. Leveraging this interpolation ratio, we generate 50 new models, each with a unique set of weights derived from these calculated ratios. The performance of these interpolated models is then meticulously assessed, capturing both loss and accuracy metrics. These metrics are plotted on a singular graph, which juxtaposes the loss and accuracy of the models across the different batch sizes, providing a visual analysis of how batch size variations influence model effectiveness.

For learning rate = 0.01

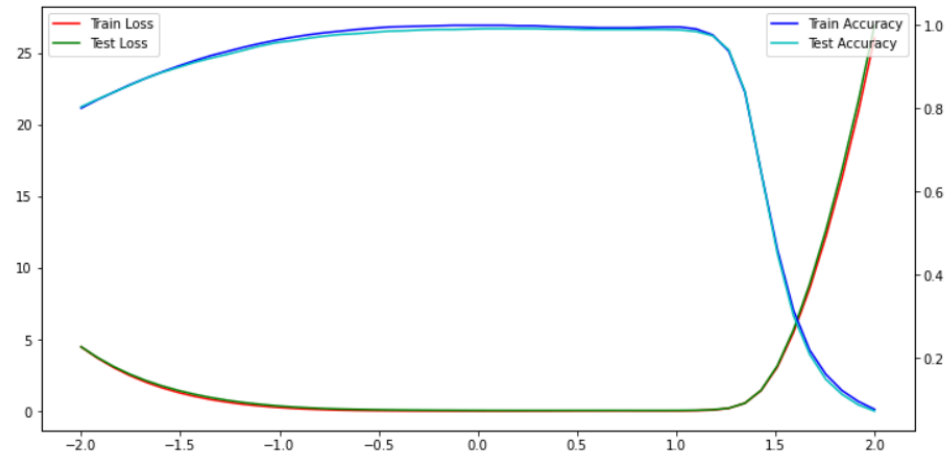


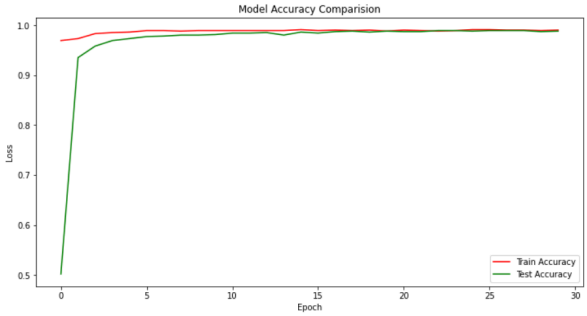
Saurabh Sharma

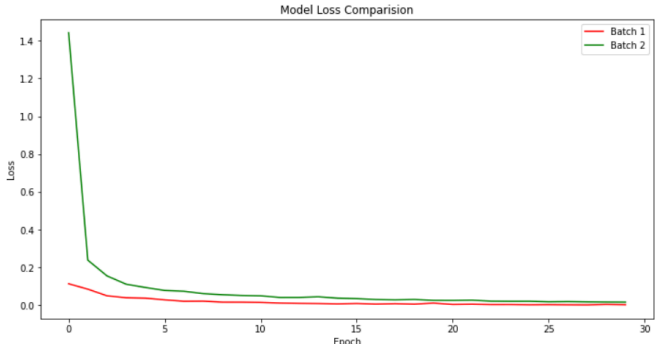




For learning rate = 0.001



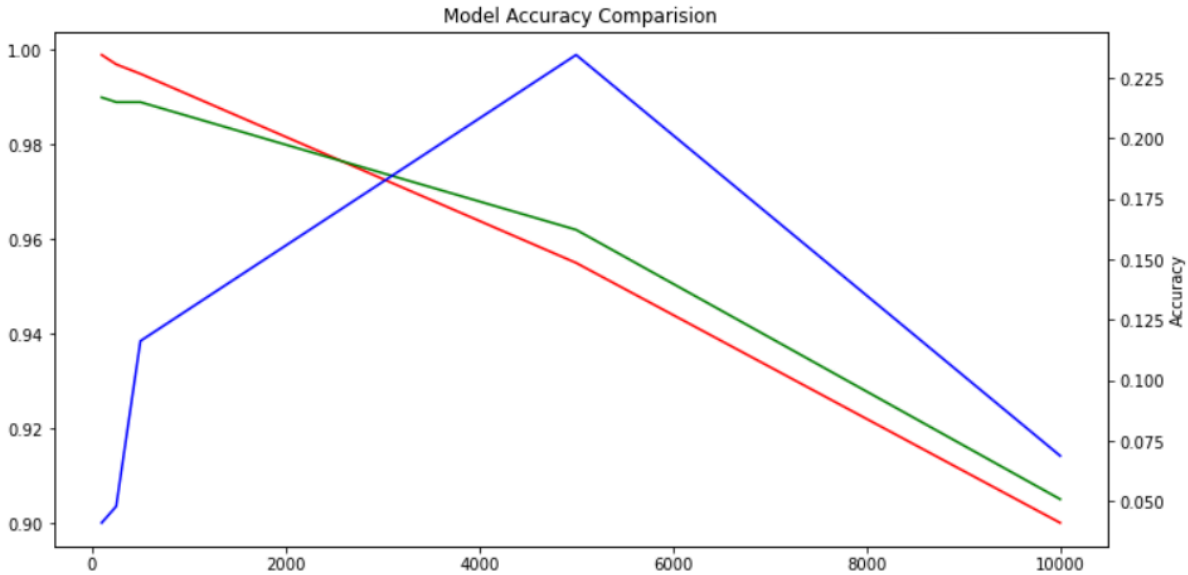




Results inference,
Graphs indicate a noticeable decline in both training and testing accuracy to a level of approximately 1.5 across models subjected to variations in learning rate. Notably, a sharp ascent in accuracy is evident from the graphs at an alpha value of 1.5, where alpha denotes the interpolation ratio, determined through the formula "(1-alpha) * batch1_param + alpha * batch2_param." This phenomenon underlines a fundamental principle within machine learning algorithms: their propensity to interpolate between data points. However, it is critical to acknowledge that an excess of parameters relative to the dataset size predisposes the models to not just interpolate but to essentially memorize the data as we described and demonstrated above in the report, illustrating the balance required in model configuration to optimize learning and generalization capabilities.

3) Flatness vs Generalization - Part 2

We are again resuing the model that we used in part 1 but this time we will only have one batch size i.e. 50.
For the graph below, x axis represents the batch_size and y axis represents the accuracy. Blue line shows the sensitivity while red and green show the accuracy and loss respectively.



The data presented in the graphs clearly illustrate a decline in sensitivity as the batch size increases, reaching a peak sensitivity at an optimal batch size of approximately 4000. Beyond this point, a further increase in batch size correlates with a decrease in accuracy and an uptick in loss values. Consequently, it can be inferred that the network's sensitivity diminishes when the batch size is expanded beyond 5000 epochs, indicating a pivotal relationship between batch size and the network's ability to accurately learn and generalize from the data.