**Homework 1**

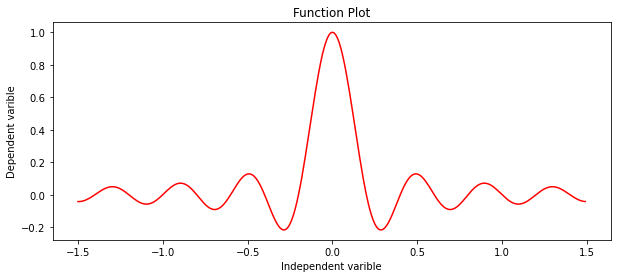
**HW 1-1 Simulate a Function**

Describe the models you use, including the number of parameters (at least two models) and the function you use.   
GitHub link: [Deep-Learning/HW1 simulate function.ipynb at master · nik1097/Deep-Learning (github.com)](https://github.com/nik1097/Deep-Learning/blob/master/Homework%201/HW1%20simulate%20function.ipynb)

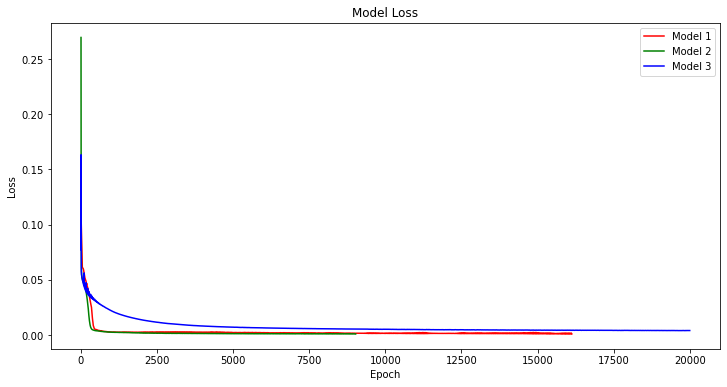
Models: I have defined 3 models that are the same as the ones shared in the requirements. A weight decay parameter has been added to regularize the models better by adding a penalty term to the cost function of a neural network which has the effect of shrinking the weights during backpropagation.

* Model 1:
  + 7 Dense Layers, 571 parameters
  + Loss Function: MSELoss
  + Optimizer: RMSProp
  + Learning Rate: 1e-3
  + Activation Function: LeakyRelu
  + Weight decay: 1e-4
* Model 2:
  + 4 Dense Layers, 572 parameters
  + Loss Function: MSELoss
  + Optimizer: RMSProp
  + Learning Rate: 1e-3
  + Activation Function: LeakyRelu
  + Weight decay: 1e-4
* Model 3:
  + 1 Dense Layer, 571 parameters
  + Loss Function: MSELoss
  + Optimizer: RMSProp
  + Learning Rate: 1e-3
  + Activation Function: LeakyRelu
  + Weight decay: 1e-4

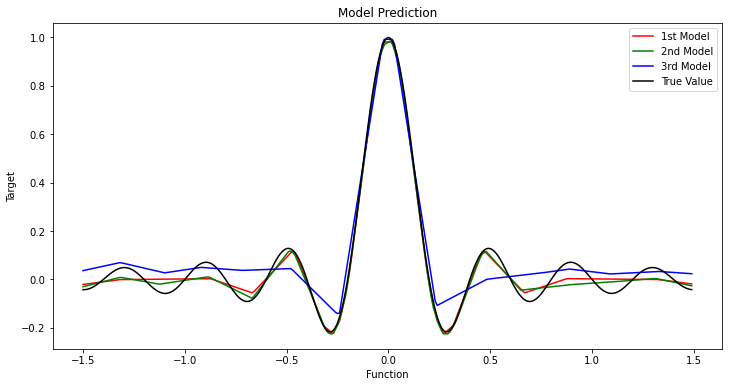
Function 1  
Function: sin(5\*pi\*x) / 5\*pi\*x  
Below is the function plot for the same function:



Simulating the Function:  
All models converge after reaching maximum number of epochs or when the model learns very slowly. Below is a graph showing the loss each of these models:

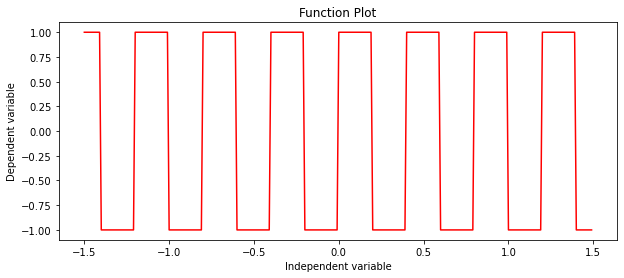


The graph below shows the Ground Truth vs the Prediction for the 3 models.

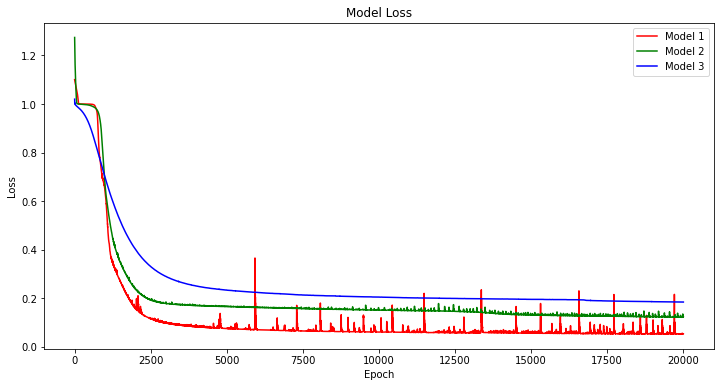


Result  
Models 1 & 2 converge quickly compared to model 3 which reaches maximum epochs before convergence. As denoted by the graph Models 1 & 2 have a lower loss value and learn the function much better than Model 3. This is a testament to the number of layers that each model possessed allowing the ones with more layers to learn faster and better.

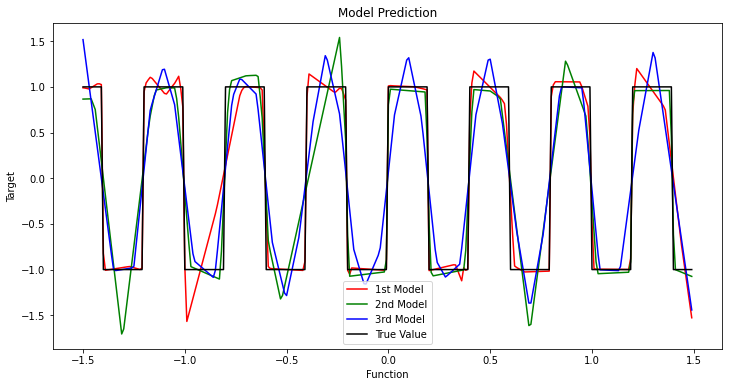
Function 2  
Function: sgn(sin(5\*pi\*x) / 5\*pi\*x)  
Below is the function plot for the same function:



Simulating the Function:  
All models converge after reaching maximum number of epochs or when the model learns very slowly. Below is a graph showing the loss each of these models:



The graph below shows the Ground Truth vs the Prediction for the 3 models.



Result  
All the models reach the maximum number of epochs before convergence as the function is difficult to learn for the 3 models. Model 1 has the lowest loss and seems to be the best learner, marginally outperforming model 2. Model 3 fails to reduce reach a low loss and fails to converge over the range of epochs. This is again testament to the fact that a neural network with more layers tends to learn better and converge quicker.

**HW 1-1 Train on Actual Tasks**GitHub link: [Deep-Learning/Hw1 comMNIST.ipynb at master · nik1097/Deep-Learning (github.com)](https://github.com/nik1097/Deep-Learning/blob/master/Homework%201/Hw1%20comMNIST.ipynb)

The below models are trained on the MNIST dataset.   
Training Set: 60,000 Testing set: 10,000

Model 1: CNN (LeNet)

* 2D convolution layer: apply a 2D max pooling -> ReLu
* 2D convolution layer: apply a 2D max pooling -> ReLu
* 2D Dense Layer: ReLu
* 2D Dense Layer: ReLu

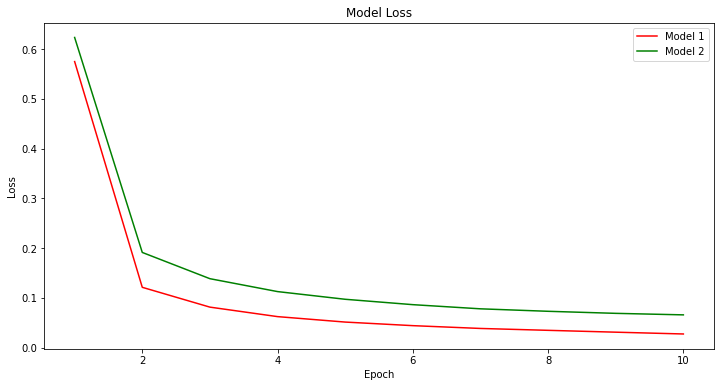
Model 2: CNN (custom)

* 2D convolution layer: ReLu
* 2D convolution layer: apply a 2D max pooling -> ReLu -> Dropout
* 2D convolution layer: apply a 2D max pooling -> ReLu -> Dropout
* 2D Dense Layer: ReLu -> Dropout
* 2D Dense Layer: Log\_softmax (Output)

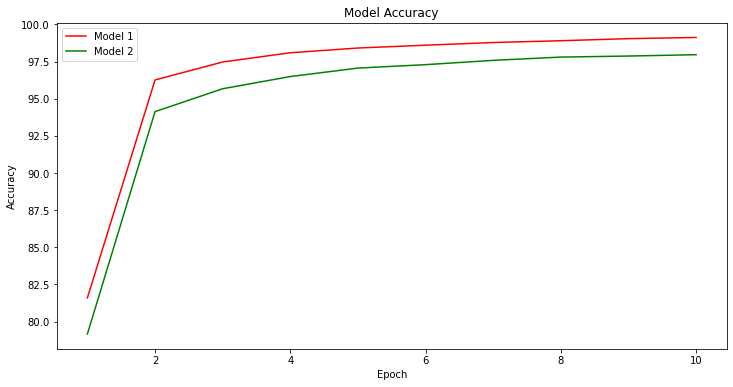
Hyperparameters

* Learning Rate: 0.01
* Momentum: 0.5
* Optimizer: Stochastic Gradient Descent
* batch\_size = 64
* epochs = 10
* Loss: Cross Entropy

Below is the training loss for Model 1 and Model 2



Below is the training accuracy for Model 1 and Model 2



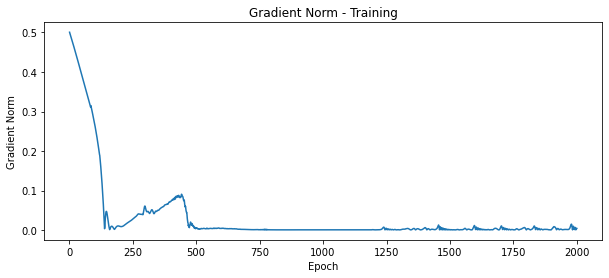
Result  
The Model 1 has a lower Loss and performs better than Model 2, the structure of Model 1 is an optimized CNN model based on the LeNet structure that exists. It greatly outperforms the custom CNN model that has been built over the range of epochs. Intuitively a similar trend can be also observed with the accuracy as the model 1 outperforms model 2 with a higher training accuracy.

**HW 1-2 Visualize the optimization process**

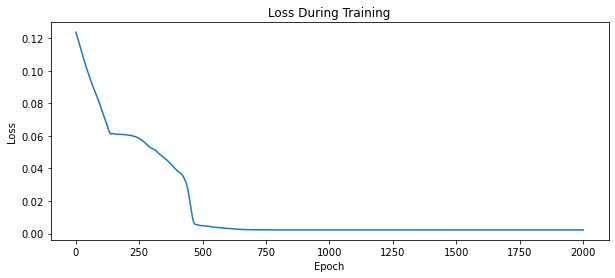
**HW 1-2 Observe gradient norm during training**GitHub Link: [Deep-Learning/HW1 grad\_norm.ipynb at master · nik1097/Deep-Learning (github.com)](https://github.com/nik1097/Deep-Learning/blob/master/Homework%201/HW1%20grad_norm.ipynb)

Thefunction sin(5\*pi\*x) / 5\*pi\*x has been reused to calculate the gradient norm and the loss. I have trained the model on epochs rather than iterations as the input to the model is already a small size.

Below is a graph for gradient norm across the epochs.



Below is a graph for loss across the epochs



ResultThe model is trained and converged. There is a slight increase in the gradient after 100 epochs which is also observed with the loss in the other graph plateauing initially and decreasing at a lower rate before finally plateauing in the end after about 500 epochs.

**HW 1-3 Can network fit random labels?**GitHub Link: [Deep-Learning/HW1 rand\_label\_fit.ipynb at master · nik1097/Deep-Learning (github.com)](https://github.com/nik1097/Deep-Learning/blob/master/Homework%201/HW1%20rand_label_fit.ipynb)

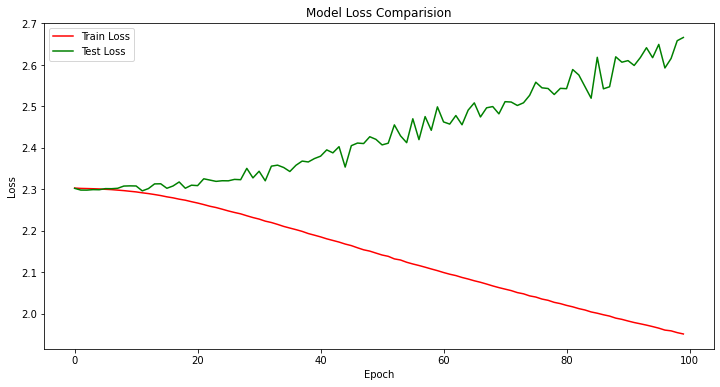
The below model is trained on the MNIST dataset.   
Training Set: 60,000 Testing set: 10,000

Model 1: CNN (LeNet)

* 2D convolution layer: apply a 2D max pooling -> ReLu
* 2D convolution layer: apply a 2D max pooling -> ReLu
* 2D Dense Layer: ReLu
* 2D Dense Layer: ReLu

Hyperparameters

* Learning Rate: 0.0001
* Optimizer: Adam
* train\_batch\_size = 100
* test\_batch\_size = 100
* epochs = 100
* loss function: Cross Entropy Loss



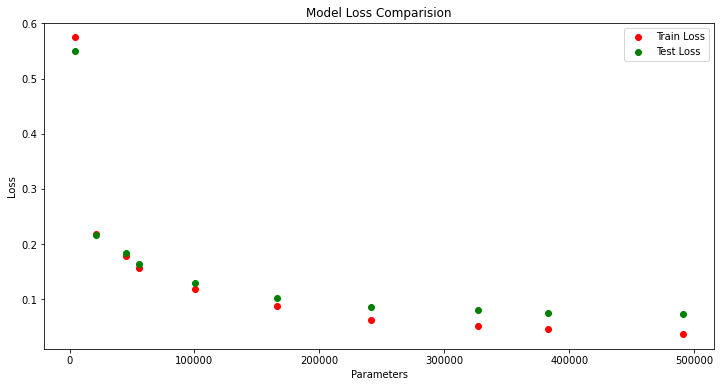
The above model is trained on random models. The training process is slow as the model must learn on random labels, it tries to memorize the labels as we move through epochs and reduce the loss. The test loss continues to increase as the epochs increase with a gradual decrease in the train loss. The above graph verifies the phenomenon and the gap between the Train and Test loss increases as we increase the number of epochs.

**HW 1-3 Number of parameters vs Generalization**GitHub Link: [Deep-Learning/HW1 param\_compare.ipynb at master · nik1097/Deep-Learning (github.com)](https://github.com/nik1097/Deep-Learning/blob/master/Homework%201/HW1%20param_compare.ipynb)

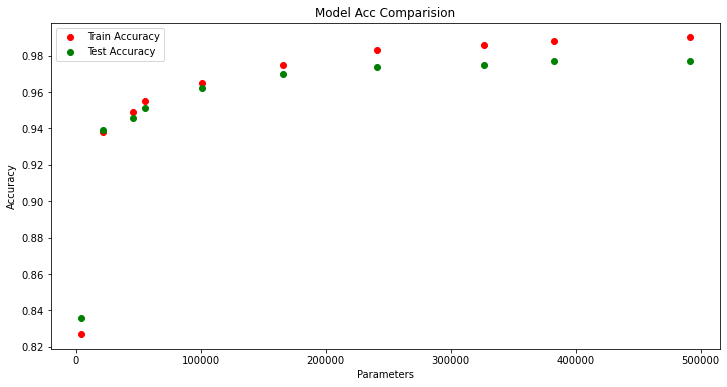
The below model is trained on the MNIST dataset.   
Training Set: 60,000 Testing set: 10,000

* 3 Dense Layers
* Loss Function: Cross Entropy Loss
* Optimizer: Adam
* Learning Rate: 1e-3
* Batch size: 50
* Activation Function: ReLu

We vary the size of the models by approximately doubling the inputs and outputs at every dense layer, this increases the number of parameters for training.  
   
Graph for Loss comparison for the various models



Graph for Accuracy comparison for the various models

****

As it can be seen from the above graphs with increase in number of parameters the difference between the train and test loss/accuracy increases. The test loss starts plateauing much before the training loss or accuracy.   
This is due to overfitting as there are more number of parameters for the model to train. Although this increases the accuracy and decreases loss in training, we must aim to reduce the difference in between train and test loss/accuracy to avoid overfitting. Due to limitations in computing power, I was unable to further increase the number of parameters, but the trend can be inferred from the above graphs.

**HW 1-3 Flatness vs Generalization  
Part 1**