CPSC 8430 (Deep Learning)- Homework 1

Harshini Gaddam

1.1: Deep vs Shallow

1.1.1: Simulate a function

Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL_HW1.1.1.ipynb

1. Three DNN models (model 0, model 1, model 2) which are presented in the ppt are used for simulating a function.

The functions are

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

$$\operatorname{sgn}(\sin(5\pi x))$$

DNN model configurations used for the above functions:

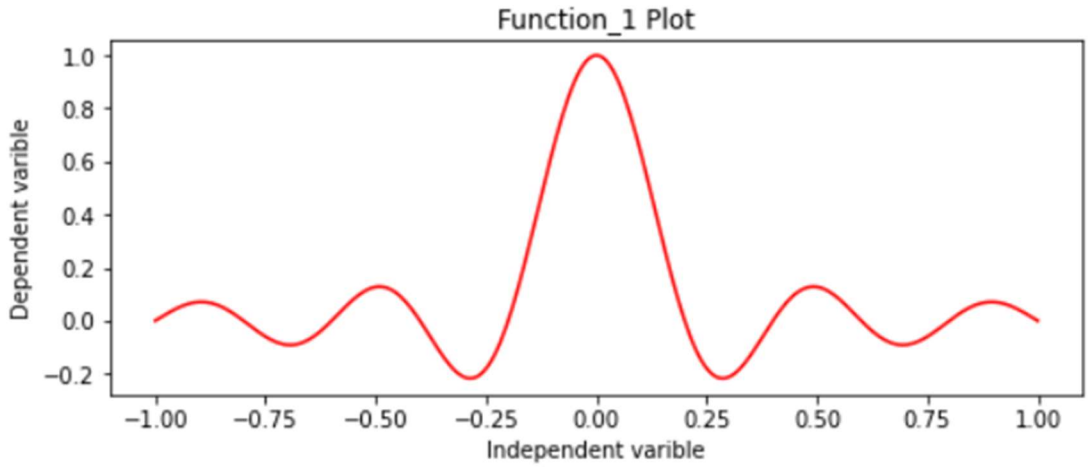
Features	**Model 0**	**Model 1**	**Model 2**
Dense Layers	7	4	1
Activation function	"Leaky_relu"	"Leaky_relu"	"Leaky_relu"
No.of parameters	571	572	571
Optimizer	"Adam"	"Adam"	"Adam"
Loss function	MSEloss	MSEloss	MSEloss
Learning rate	0.0012	0.0012	0.0011
Weight decay	1e-4	1e-4	1e-4

2. We can see that model is not overfitted because of the regularization technique "Weight decay". It also reduces the complexity and improves the generalization performance.

3. Max epoch set used is 20000(as stated in ppt), this value is used to train all the models.
4. Train function completes the execution after reaching the following conditions;
    1. max epoch reached
    2. The loss is virtually equal to zero, indicating that the model has converged (stopped learning) (loss stops dropping further).

Function-1:

Plot of the function $\frac{\sin(5\pi x)}{5\pi x}$

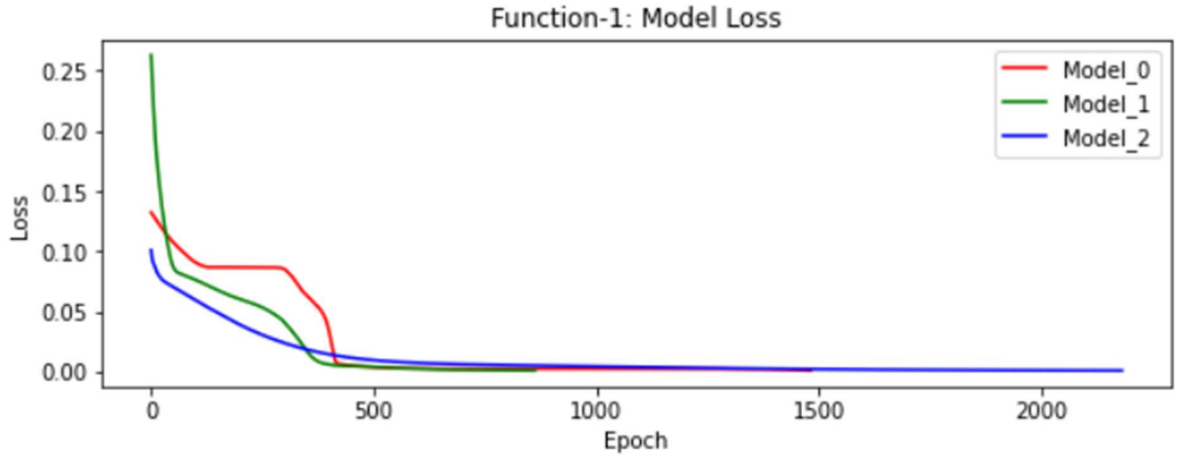


Max epoch =20000
Model 0 with 7 hidden layers has reached convergence for loss=0.001 and at epoch =1481

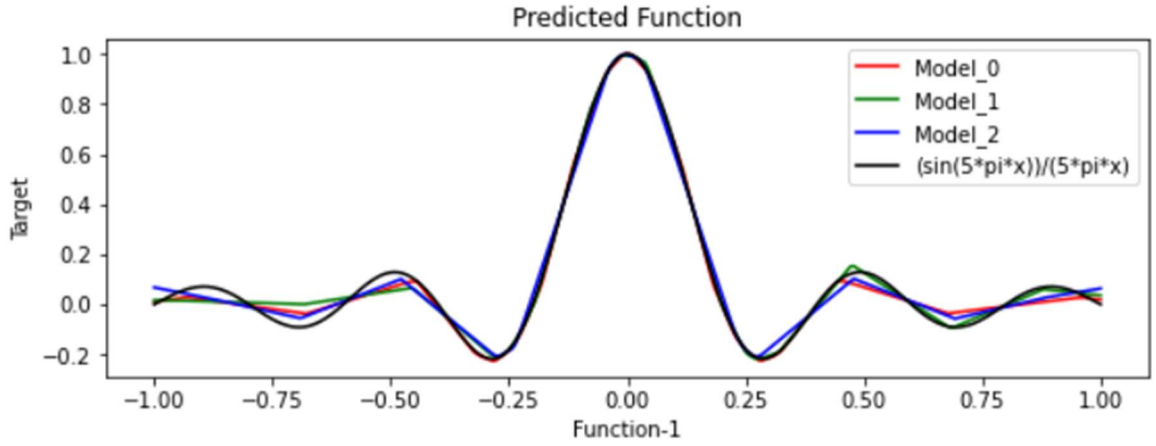
Model 1 with 4 hidden layers has reached convergence for loss=0.001 and at epoch =862

Model 2 with only 1 hidden layer has reached convergence for loss=0.001 and at epoch =2179

The plot (epoch vs Loss) visualizes the loss against no.of epochs of all the 3 models of function 1



The following graph visualizes the model prediction by all the 3 models of function 1.

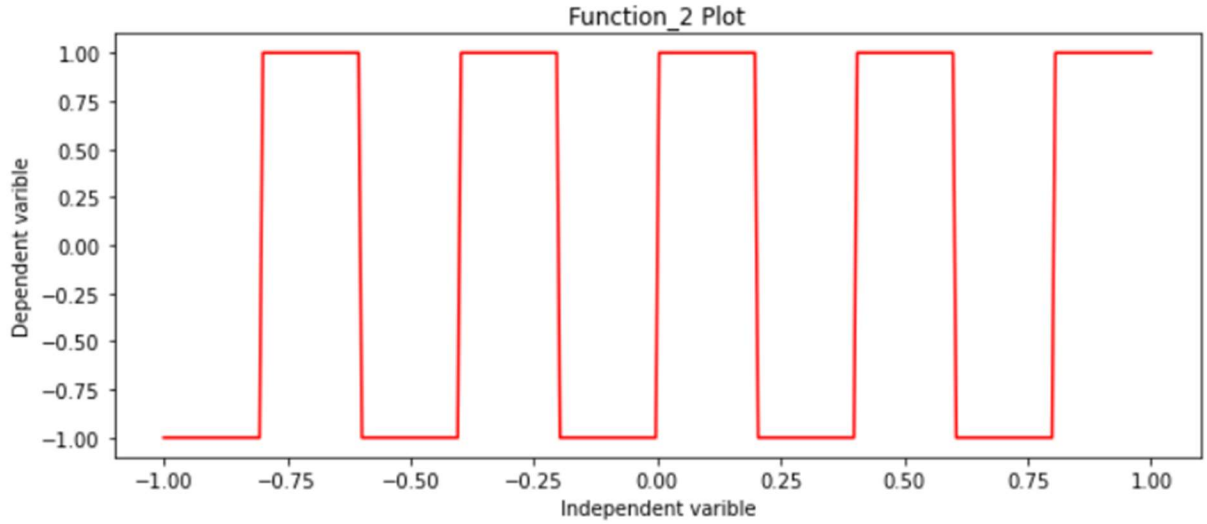


Observation:

Model 0 & 1 has reached convergence (at epoch=1481, 862) faster than model 2 (at epoch=2179). Since model 0 has more layers, it can be said that it has more capacity and it is computationally more efficient. Since model 2 has only 1 hidden layer, it takes more epochs to converge.

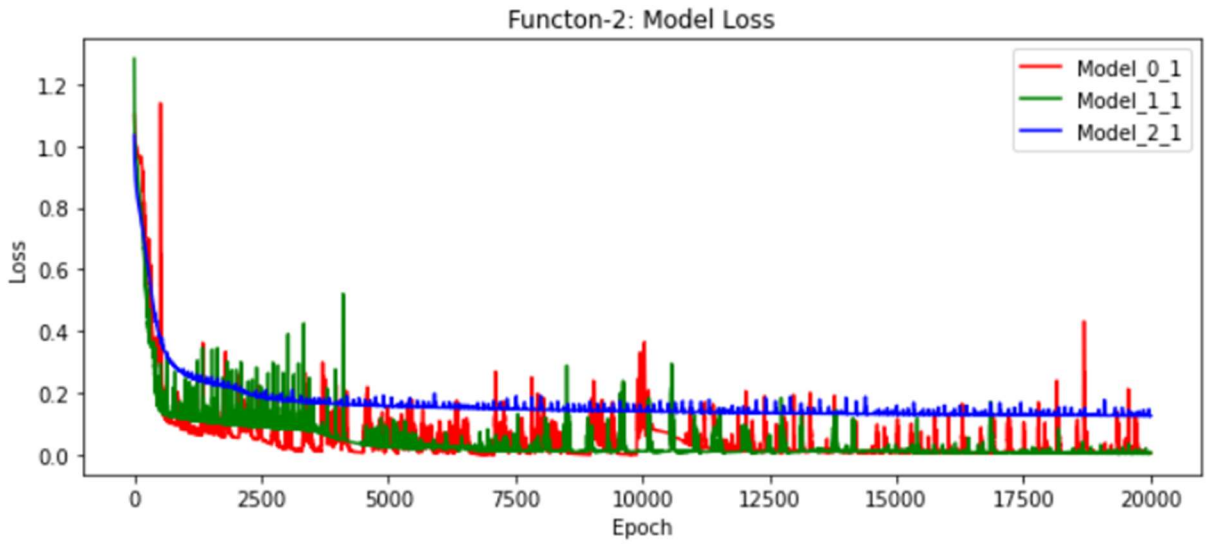
Function-2:

Plot of the function $\operatorname{sgn}(\sin(5\pi x))$

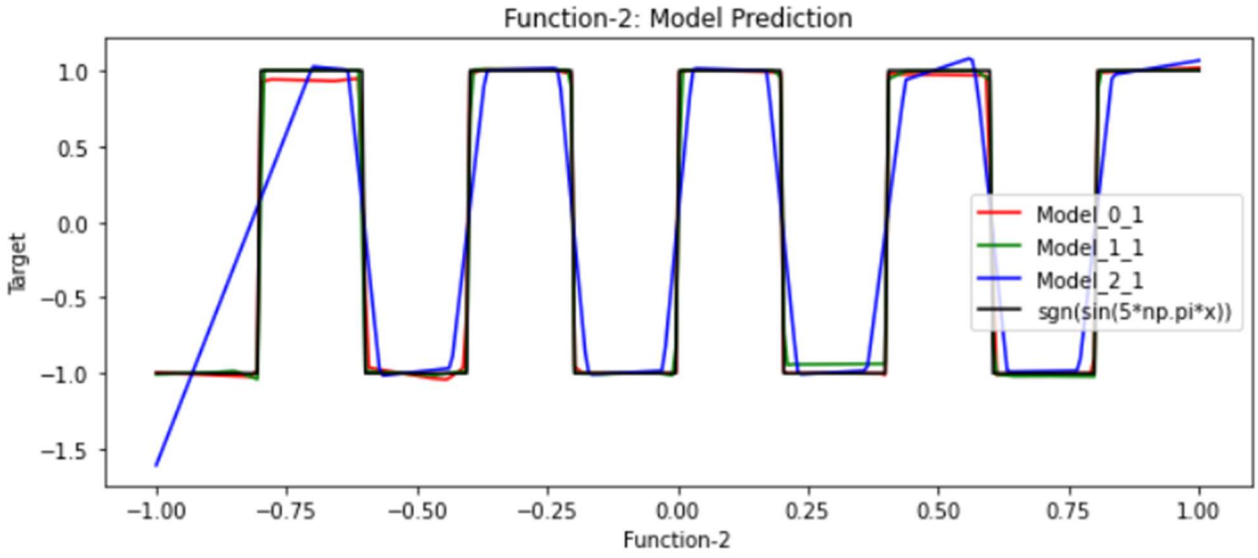


1. All the configurations are used same as function1 but hyper-parameter learning rate is set to 0.009
2. All the Models has reached the convergence at epoch=20000

The plot (epoch vs Loss) visualizes the loss against no.of epochs of all the 3 models of function 2



The following graph visualizes the model prediction by all the 3 models of function 2.



Observation:

All the 3 models have reached convergence at epochs=20000. Prediction by model 0 and model 1 is almost like the original function. But the prediction by model 2 is not similar to the original function and it is distorted. This can be said because model 2 has only 1 hidden layer. For the models to reach the convergence with the less no.of epochs, they need to have more no of layers.

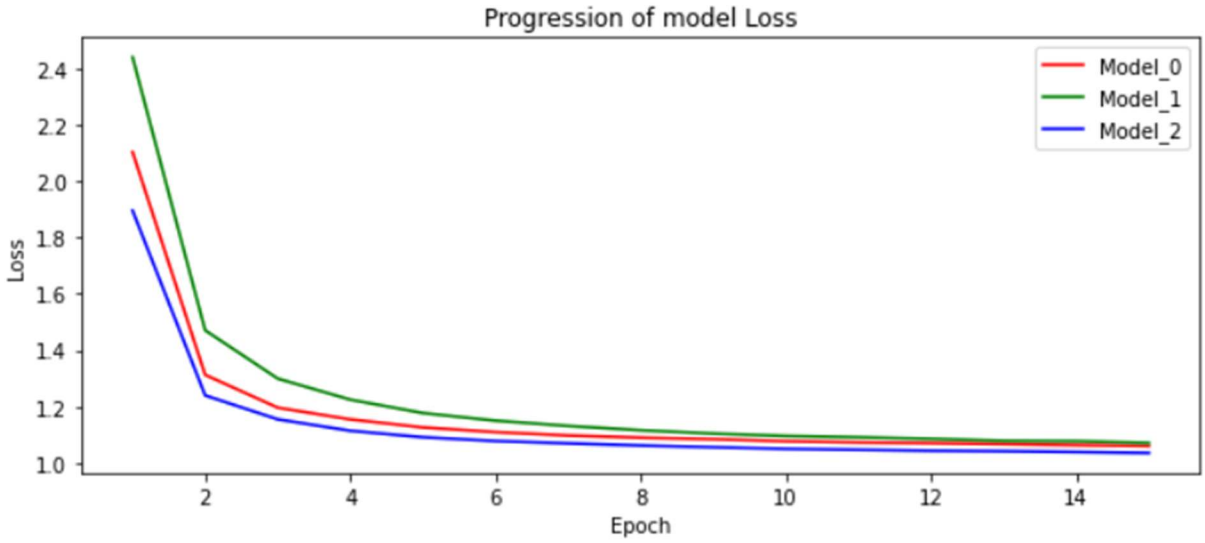
1.1.2: Train on Actual Tasks

Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL_HW_%201.1.2.ipynb

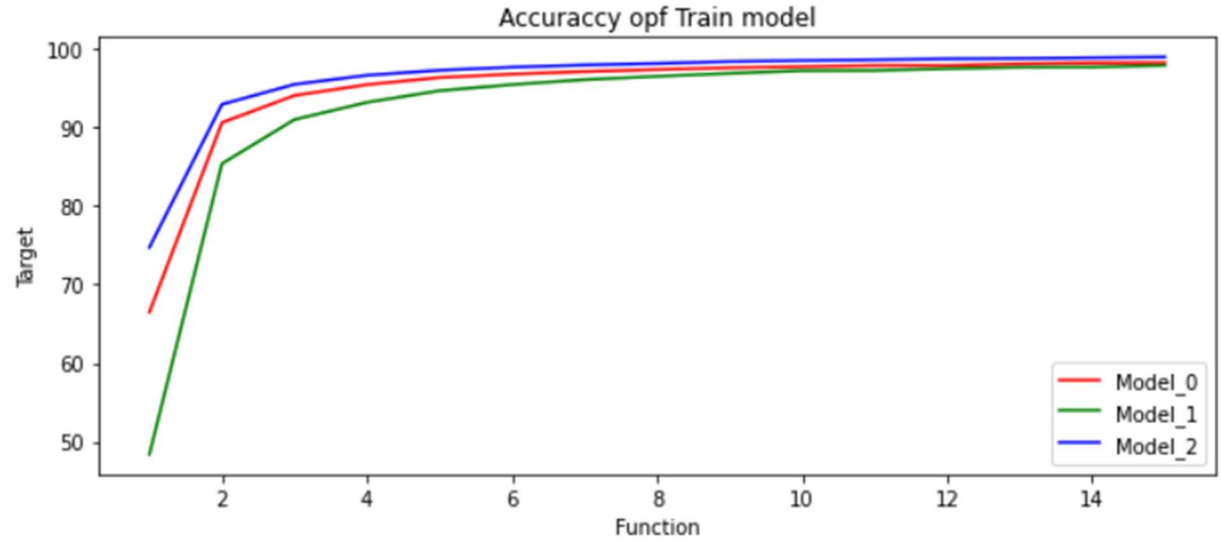
1. MNIST dataset is used to train. 3 CNN models are created with the following configurations.

Features	CNN 1	CNN 2	CNN 3
No.of convolution layers, Kernel size	2, 4	2,4	2,4
Pool size, Strides	2,2	2,2	2,2
Dense layers	2	4	1
Activation function	“relu”(Rectilinear Linear Unit)	“relu”	“relu”
No.of parameters	25550	25570	25621
Loss function	Cross Entropy loss	Cross Entropy loss	Cross entropy loss
Optimizer function	Adam	Adam	Adam
Learning Rate	0.0001	0.0001	0.0001
Weight decay	1e-4	1e-4	1e-4
Dropout	0.25	0.25	0.25

The following plot visualizes the training loss of all the models against no.of epochs.



The following plot is the training accuracy of all the models when plotted function against target.



Observation:

Model1:
No.of parameters=25550, loss=0.0919, Accuracy=98.191%
Model2:
No.of parameters=25570, loss=0.0842, Accuracy=97.85%
Model3:
No.of parameters=25621, loss=0.0247, Accuracy=98.911%

Though model3 has only 1 dense layer, it performed better than the other 2 models with more no.of dense layers. Model3 has less loss and accuracy is also higher for model3.

Test Accuracy:
Model1: 98.71%
Model2: 98.%
Model3: 98.8%

Even the test accuracy of model3 is higher than than the other 2 models.

1.2: Optimization

1.2.1: Visualize the optimization process

Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL_HW_1.2.1.ipynb

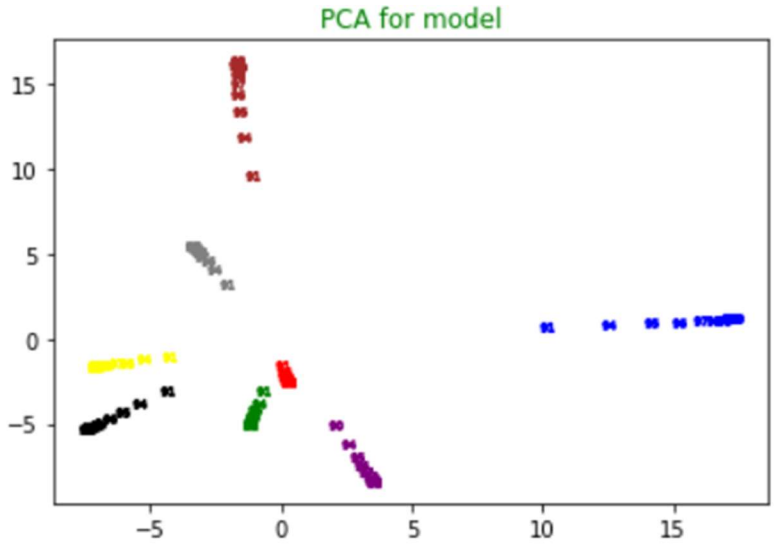
1. To visualize the optimization process, the following experimental settings are used.
MNIST dates and DNN model is used.
Dense layer-1
Activation Function:“relu”
Total no. of parameters: 397510
Loss Function: “CrossEntropyLoss”
Optimizer Function: “Adam”
Hyperparameters:
    Learning Rate: 0.0004
    Weight Decay: 1e-4

2. Model is trained for 8 times while collecting the weights for each epoch throughout the course of 45 epochs.
3. Weight is sorted for every third epoch and PCA is used to reduce the dimension by 2.

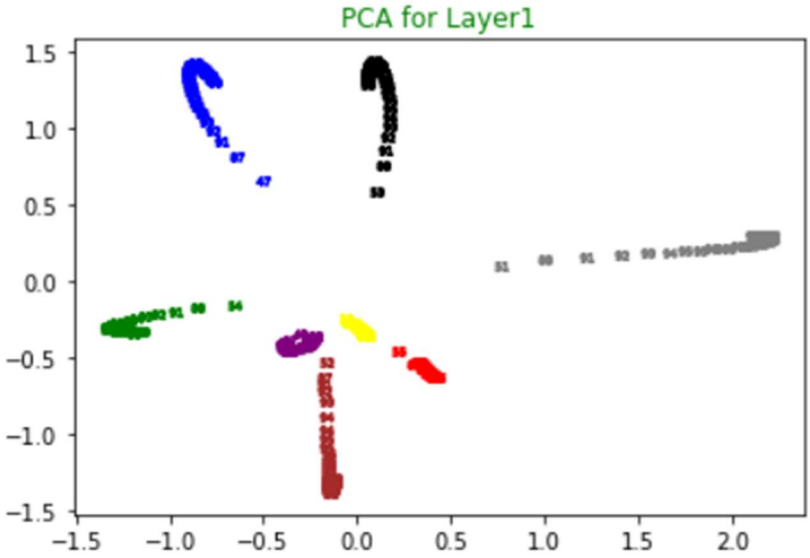
	x	y	Epoch	Iteration	Acc	Loss
0	0.057727	-1.543150	2	0	91.693788	0.293345
1	0.085329	-1.900704	5	0	94.463370	0.192326
2	0.125524	-2.129528	8	0	95.860968	0.142754
3	0.166914	-2.291441	11	0	96.818877	0.110240
4	0.195779	-2.398232	14	0	97.555905	0.086601
...	...	...	...	...	...	...
115	-3.413284	5.513035	32	7	99.403393	0.027352
116	-3.409348	5.499432	35	7	99.588629	0.023444
117	-3.392471	5.471549	38	7	99.637851	0.020139
118	-3.372816	5.437819	41	7	99.757131	0.016937
119	-3.350400	5.395371	44	7	99.821089	0.014929

120 rows × 6 columns

Plotting the weights of an entire model



Plotting the weights of 1 layer



Observation:

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of datasets, and improves the interpretability while minimizing information loss. These are done by creating new uncorrelated variables that successively maximize variance. Initially before PCA there were 418060 weights for each model, then PCA is performed and the graphs are plotted for the entire model and for 1 layer.

1.2.2: Observe gradient norm during training

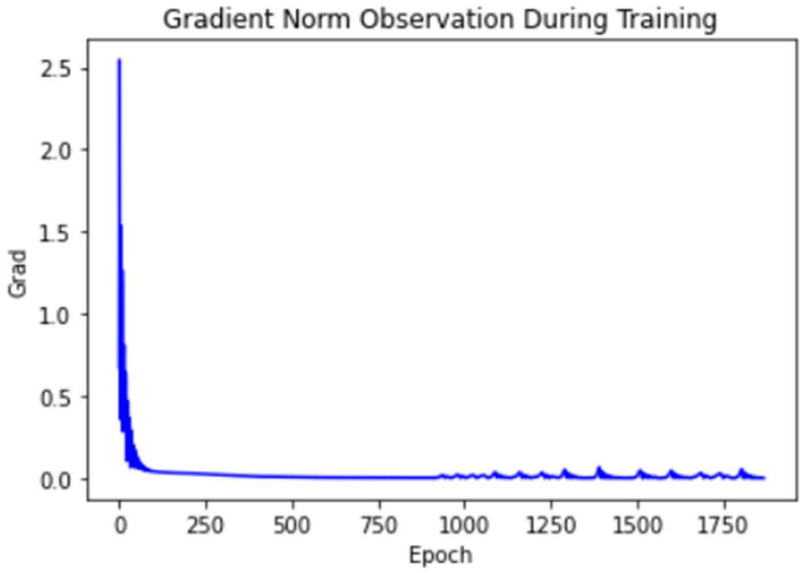
Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_1.2.2.ipynb

Function 1 $\frac{\sin(5\pi x)}{5\pi x}$, MNIST dataset and a DNN model are used.

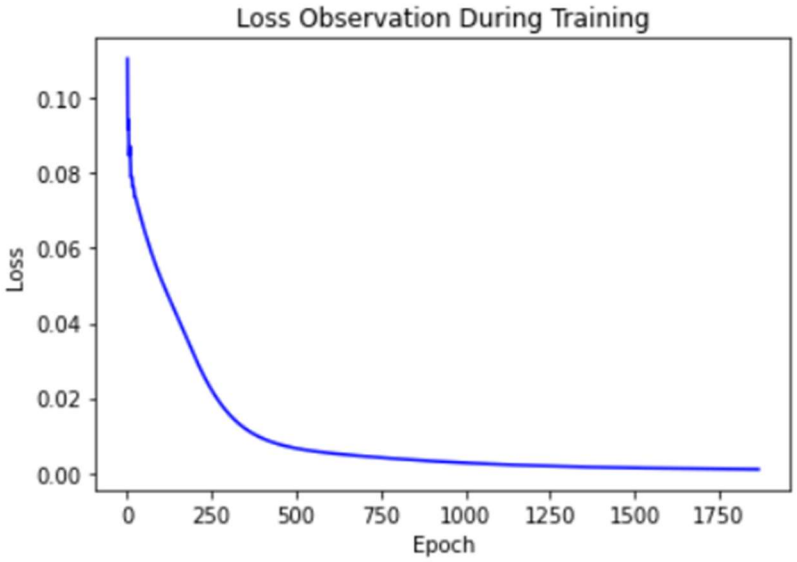
Max epoch=2500

Learning rate: 1e-3

Plot that contains gradient norm to iterations



Plot that contains loss to iterations.



Observations:

Gradient norm is used to balance the training in the model, by dynamically tuning gradient magnitudes. After training the model over 1800 epochs, convergence reached for loss=0.0009999699. Random increase or decrease in gradient norm is observed. Till 1000 epochs, continuous decrease is observed and increased for 1100 epochs. No continuous trend is seen and for epoch 1800, grad-norm value is “0.04609107525235673”. Thee is continuous decrement in loss and at certain point is seems slightly constant. Finally the convergence loss is “0.0009999699”.

1.2.3: What happens when gradient is almost zero?

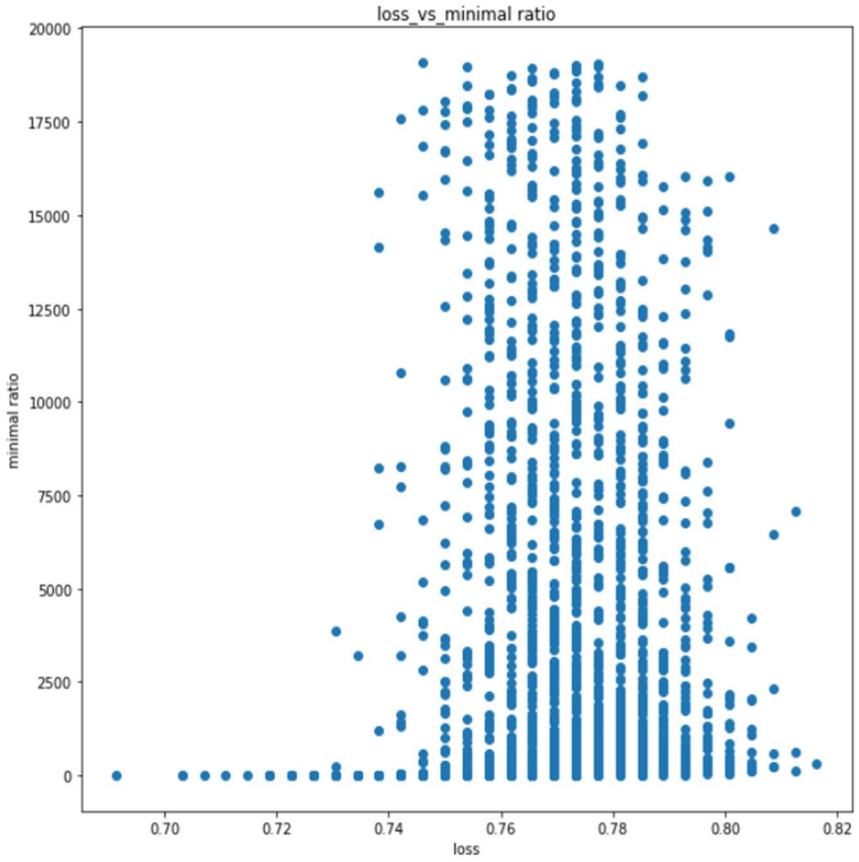
Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_1.2.3.ipynb

Function 1 $\frac{\sin(5\pi x)}{5\pi x}$, MNIST dataset and a DNN model is used.

Max epochs=2000

After training the model for 100 times, loss is never zero for the model, and the minimum gradient obtained is 1.3272, ratio: 0.6914

The following is the plot between the minimal ratio and the loss.



Observation:

When gradient norm is zero, the model stops learning and it will not be able to memorize and the model reached local minima.

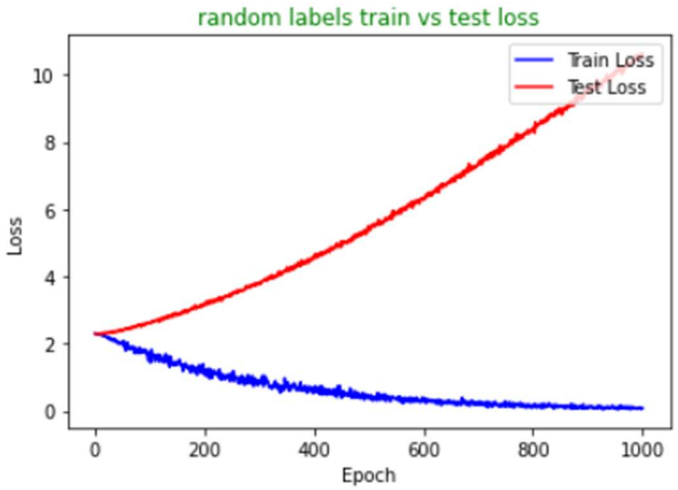
1.3: Generalization

1.3.1: Can network fit random labels?

Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_1.3.1.ipynb

MNIST dataset from torchvision is taken, with the training dataset size of 60000, testing dataset size of 10000.
Environment setting:
1 Dense layer
Activation Function : "relu"
Total no. of parameters: 397510
Loss Function: "CrossEntropyLoss"
Optimizer Function: "Adam"
Hyperparameters- Learning Rate: 0.0001

Here is the plot of epoch vs loss obtained by training the model with the train and test conditions.



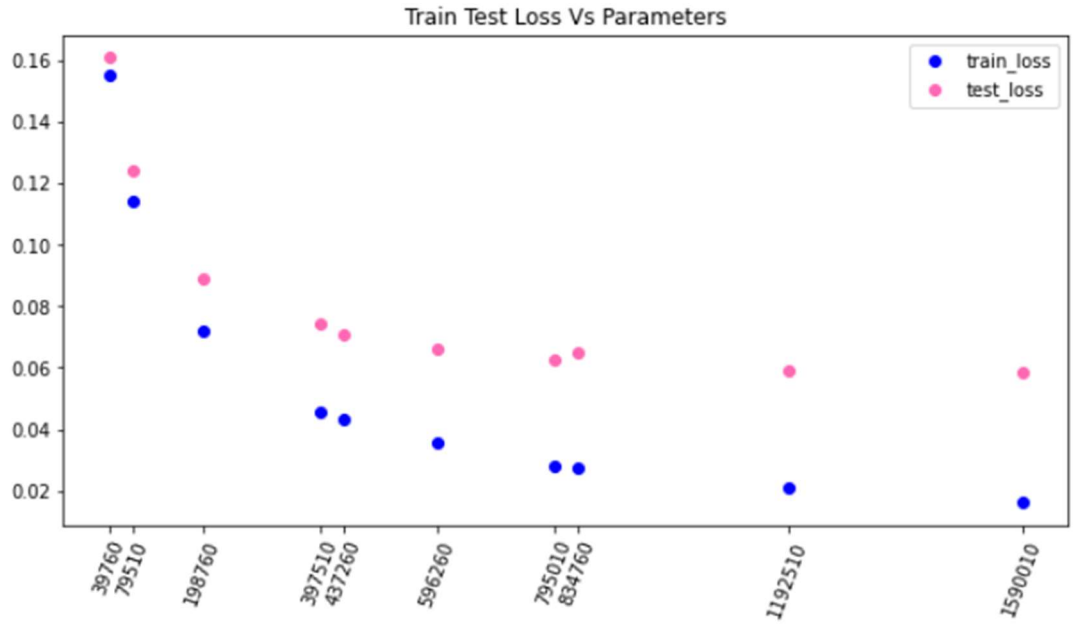
Observation:
It can be seen that model can learn and memorize the data to minimize the loss. No correlation between training and testing labels is seen. Loss decrement in the training data is seen and as a result, loss in testing data is increased. After 1000 epochs, train loss of 0.0565, test loss of 10.529 and the network accuracy on the test images is 10.66% is observed.

1.3.2: Number of parameters v.s. Generalization

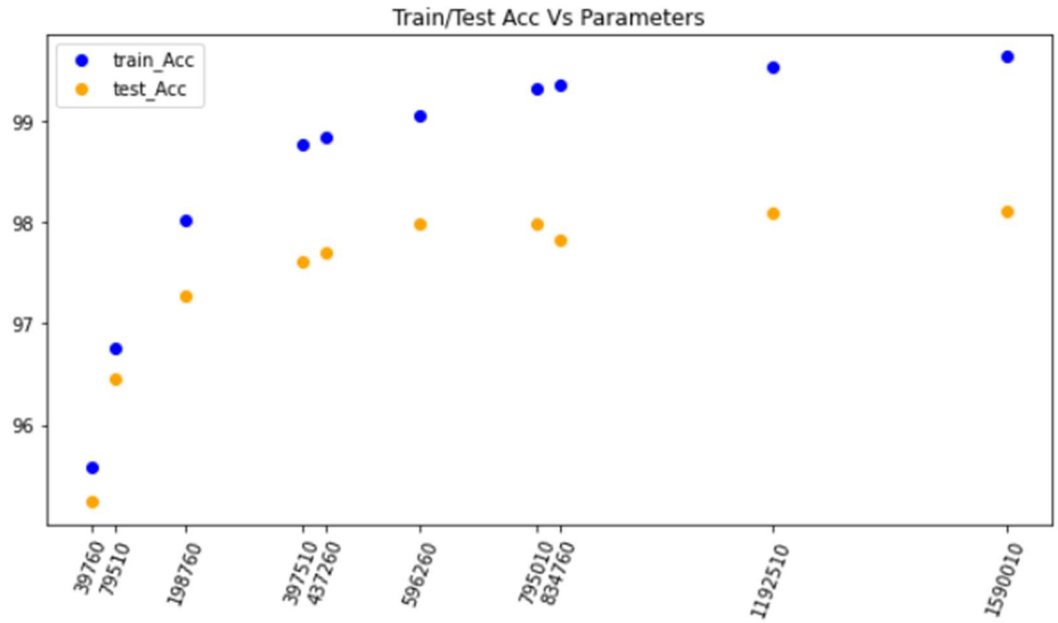
Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_1.3.2.ipynb

MNIST dataset is used and 10 similar structured CNN models are used with different number of parameters. No.of parameters in the models are 39760, 79510, 198760, 397510, 437260, 596260, 795010, 834760, 1192510, 1590010.
Environment setting:
No.of Convolution Layers: 2 with Kernel size =4
Max pooling with pool_size =2 and Strides =2
No.of dense layer = 2
ActivationFunction:“relu”
Total no. of parameters: 25550
Loss Function: “CrossEntropyLoss”
Optimizer Function: “Adam”
Hyperparameters: Learning Rate: 0. 0001, Dropout = 0.25

Plot that compares training and testing loss.



Plot that compares training and testing accuracy.



Observation:

The above plots are plotted with respect to the no. of parameters, it can be seen that as parameters increase, the loss of training and testing data decreases and the accuracy increases. The model did not generalize and did not fit closely to the training dataset and overfitting is seen. The model is said to be the best if it has the least loss and best accuracy and also less overfitting. Overfitting is the gap that is observed between training and testing data. If overfitting decreases, the performance of the model increases.

1.3.3: Flatness vs Generalization

Part 1

Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_%201.3.3.1.ipynb

Two models M1 and M2 are trained with the batch_size=64, 1024.

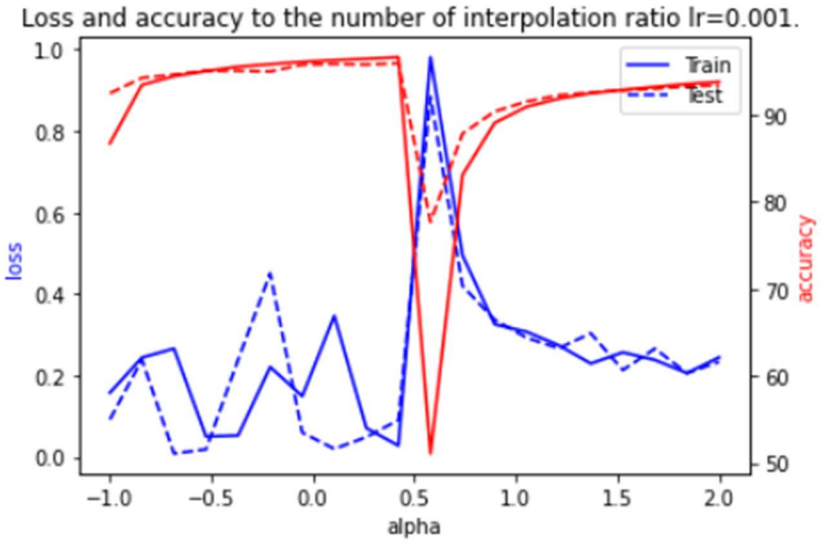
MNIST dataset is used and 2 DNN models are created.

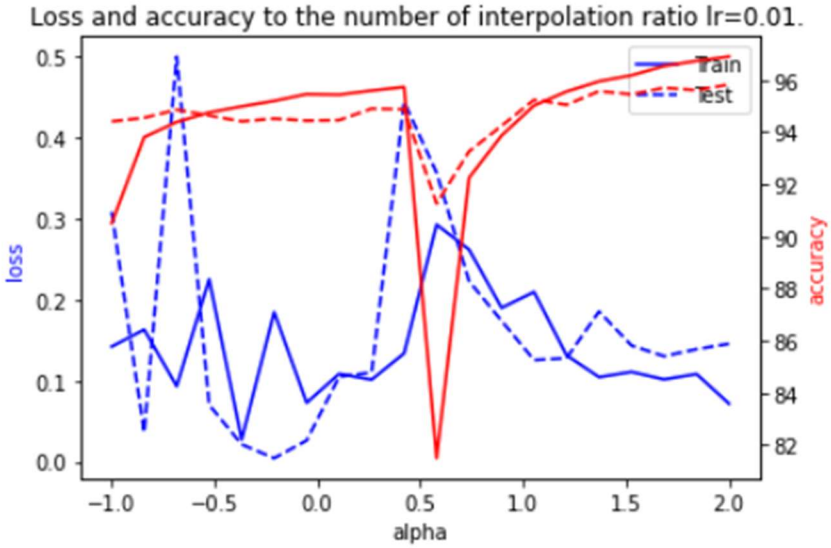
Model M1

Specifications:
1 Dense layer
ActivationFunction:“relu”
Total no. of parameters: 16950
Loss Function: “CrossEntropyLoss”
Optimizer Function: “Adam”
Max Epoch: 10
Hyperparameters: Learning Rate: 0.001 Weight decay: 1e-4

Model M2:
MNIST dataset is used and 2 DNN models are created.
Batch_size= 1024
Learning rate: 0.001
Total no.of parameters: 16950
Optimizer: Adam
Max_epochs: 10

Plotting of training and testing, loss and accuracy to the number of interpolation ratio(alpha).
With learning ratio: 0.001, 0.01





Observations:

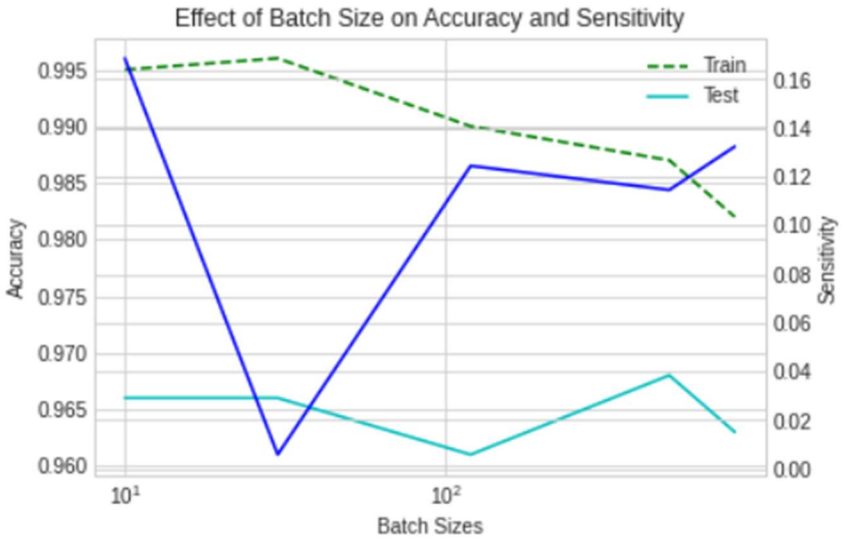
Alpha is the interpolation ratio of theta 1 and theta 2 which are model parameters. Interpolation is used to determine the unknown points that lie in between known data. Loss is cross entropy loss. It is a metric which is used to measure the betterment of the model and model weights can be adjusted during training using cross-entropy loss. The model is said to be perfect if it has 0 cross-entropy. The 2 models are trained with different learning rates. Loss is being reduced and accuracy is being increased.

1.3.3.2
Part 2

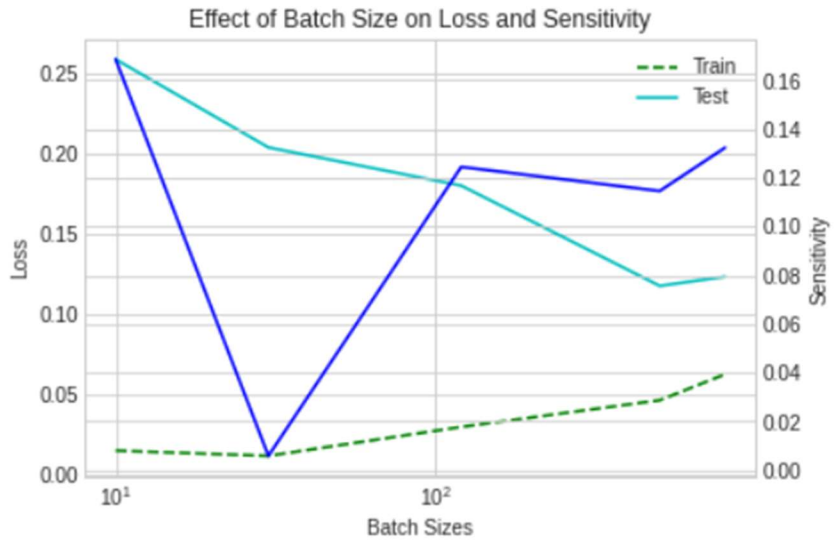
Code link: https://github.com/Harshini-Gaddam/Cpsc-8430-HW1-Harshini/blob/main/DL-HW_%201.3.3.2.ipynb

MNIST dataset, DNN models are used.
Batch sizes of the 5 models are [10, 30, 120, 500, 800]
All the models have 20715 parameters.
Max_epochs: 50
Loss Function: "CrossEntropyLoss"
Optimizer Function: "Adam"
All the models have the same design specifications like learning rate 0.001, epochs.

Plot of training, testing dataset, Accuracy, sensitivity and batch sizes.



Plot of training, testing dataset, loss, sensitivity and batch sizes.



Observations:

To analyze the impact of batch size on loss, accuracy and sensitivity the above graphs of train and test are plotted. It is seen that as the batch size increases, performance decreases(most of the times) and at some batch sizes, it increased. It can be said that the model should have the least batch size to learn faster and perform efficiently. Sensitivity is decreased when batch size is increased. That means the model is less sensitive to the data changes. But sensitivity and accuracy should be properly balanced for the model to be efficient and to produce accurate output.