CPSC 8430: Deep Learning Homework 1

Mary Damilola Aiyetigbo

(https://github.com/dimky01/8430-Deep-Learning)

This homework is divided into three parts:

1. Training Deep network and shallow networks with the same number of parameters
    a. Trained two (2) simulated functions on 3 different models
    b. Trained the different models using the MNIST dataset
2. Optimization
    a. Visualized optimization process using PCA
    b. Observed gradient norm during training with both MNIST dataset and simulated function
    c. Observed when gradient is almost zero
3. Generalization
    a. Can network fit random labels
    b. Number of parameters vs. generalization
    c. Flatness vs. generalization

PART 1: Deep vs. Shallow

1.1 Simulate a function

I developed 3 different models of Deep Neural networks (DNN) with the same number of parameters. The fir[st] and thi[rd] models had 571 parameters, while the 2[nd] model had 572 parameters. These models were trained on the following two (2) different functions:

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

Model1:

- 7 hidden layers
- Layer1: Neurons = 5
- Layer2: Neurons = 10
- Layer3: Neurons = 10
- Layer4: Neurons = 10
- Layer5: Neurons = 10
- Layer6: Neurons = 10
- Layer7: Neurons = 5
- # Parameters = 572
- Optimizer = Adam
- Loss Function = MSE Loss
- Hyperparameters:
    - Learning Rate = 0.001
    - Weight Decay = 0.0001

Model2:

- 5 hidden layers
- Layer1: Neurons = 10
- Layer2: Neurons = 18
- Layer3: Neurons = 15
- Layer4: Neurons = 14
- # Parameters = 571
- Optimizer = Adam
- Loss Function = MSE Loss
- Hyperparameters:
    - Learning Rate = 0.001
    - Weight Decay = 0.0001

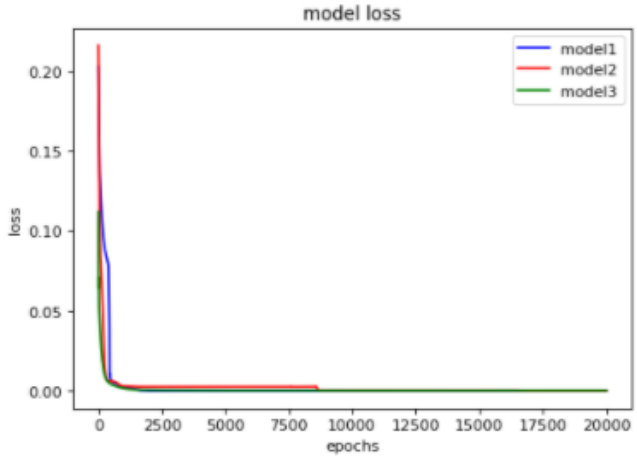
Model3:

- 1 hidden layer
- Layer1: Neurons = 190
- # Parameters = 571
- Optimizer = Adam
- Loss Function = MSE Loss
- Hyperparameters:
    - Learning Rate = 0.001
    - Weight Decay = 0.0001

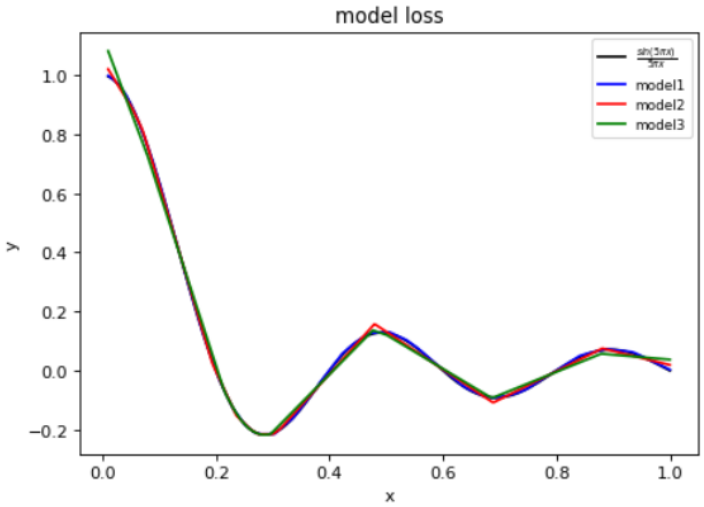
All the 3 models were trained for 20,000 epochs, and weight decay was used as a regularization technique to prevent overfitting during training.

- **Function 1:** $\frac{\sin(5\pi x)}{5\pi x}$

The graph below shows the plot of model loss versus the number of epochs for all the three models



The graph below shows the model prediction for all three models against the ground truth

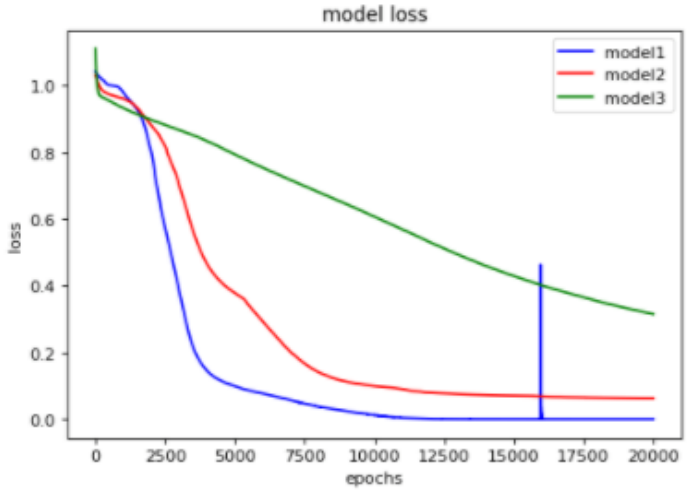


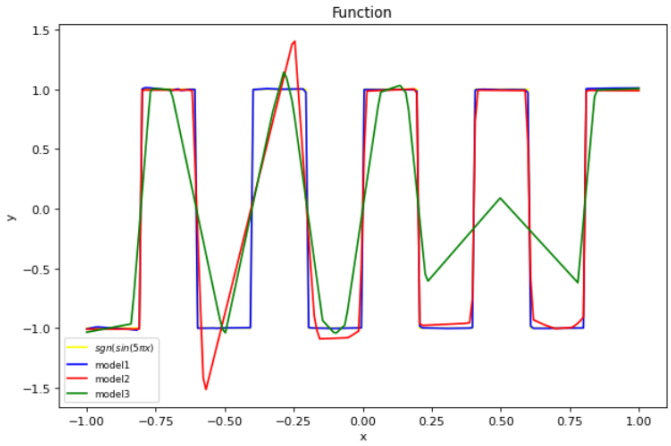
Conclusion:

- **Function 2:** $sgn(\sin(5\pi x))$

For this function, the learning rate was updated to 0.0001, and the models were trained for 20,000 epochs.

The graph below shows the plot of model loss versus the number of epochs for all the three models





From the model loss plot above, model 1 converged faster than the other models, and it had the lowest loss, followed by model2. While model 3 could not reach convergence after 20,000 epochs. Based on the performance of the models when plotted against the ground truth, model 1 fit the ground truth labels very closely than the other 2 models.

1.2 Train on Actual Task: MNIST

I trained 3 DNN models and 3 CNN models on the MNIST dataset for this task. Each model has a different number of parameters and different layers.

DNN MODELS

All the 3 DNN models were trained for 40 epochs with RELU activation functions for all the hidden units, and the SoftMax activation function was used for the outer (logits) layer. The outer layer of each model has 10

logits, and negative log-likelihood (nn.NLLLoss()) was used as a loss function. ReLu activation function was used for each layer of the network

DNN 1:

- 4 hidden layers
- Hidden Layer1: Neurons = 128
- Hidden Layer2: Neurons = 64
- Hidden Layer3: Neurons = 32
- Hidden Layer4: Neurons = 16
- # Parameters = 111514
- Optimizer = Adam
- Learning Rate = 0.001

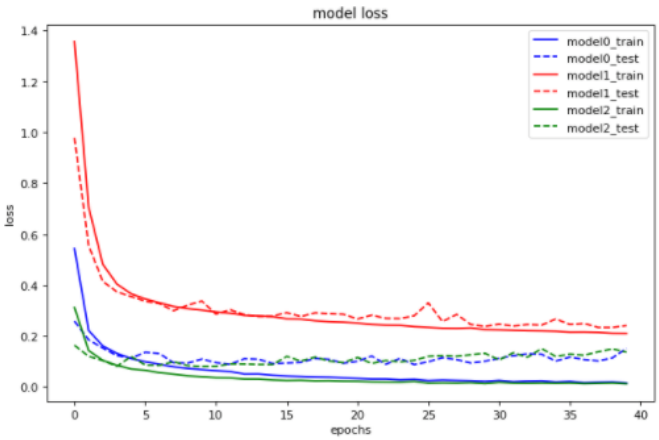
DNN 2:

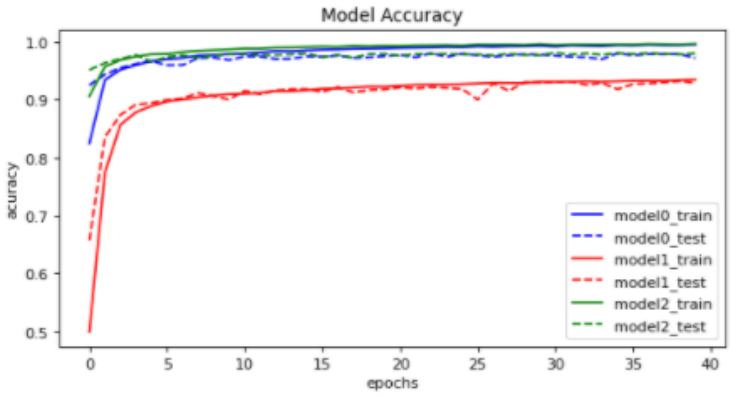
- 7 hidden layers
- Hidden Layer1: Neurons = 16
- Hidden Layer2: Neurons = 32
- Hidden Layer3: Neurons = 32
- Hidden Layer4: Neurons = 32
- Hidden Layer5: Neurons = 32
- Hidden Layer6: Neurons = 32
- Hidden Layer7: Neurons = 16
- # Parameters = 18026
- Optimizer = Adam
- Learning Rate = 0.001

DNN 3:

- 1 hidden layer
- Hidden Layer1: Neurons = 512
- # Parameters = 407050
- Optimizer = Adam
- Learning Rate = 0.001

Below is the plot of the training loss and the test loss for all the 3 models against the epochs





With only one hidden layer, the third model (Model 3) has the best model performance with the highest training and test accuracy and the lowest training loss with the highest convergence. It can be concluded that the model had good performance because it has the highest number of parameters (407,050).

Below is the training accuracy and training loss of the models

- Model1: Train Acc: 99.25% Train Loss: 0.02325
- Model2: Train Acc: 93.13% Train Loss: 0.22202
- Model3: Train Acc: 99.51% Train Loss: 0.01427

Below is the test accuracy and test loss of all the 3 models

- Model1: Test Acc: 97.50% Test Loss: 0.1066
- Model2: Test Acc: 92.09% Test Loss: 0.2561
- Model3: Test Acc: 97.84% Test Loss: 0.1203

CNN MODELS

All the 3 CNN models were trained for 100 epochs with RELU activation functions for all the hidden units, and the SoftMax activation function was used for the outer layer. The outer layer of each model has 10 logits/classes, and the NLLLoss loss function was used as a loss function. In order to avoid overfitting, dropout 0.2 was used in the fully connected layers.

CNN 1:

- Convolution layer1: filter=8; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2) and stride=(2x2)

- Convolution layer2: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- FC1: Neurons=512; Dropout=0.2
- # Parameters = 408,298
- Optimizer = Adam
- Learning Rate = 0.0001
- Weight Decay = 0.003

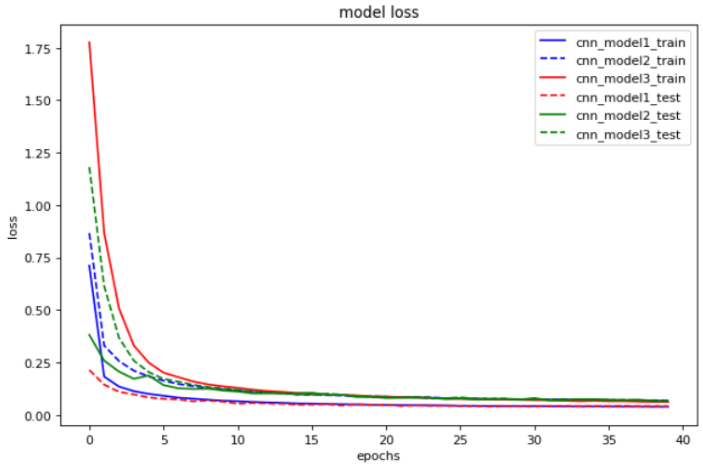
CNN 2:

- Convolution layer1: filter=16; kernel=(4x4); stride=(2x2) and padding=(2x2)
- Max pooling: kernel=(2x2); stride=(3x3); padding=(1x1)
- Convolution layer2: filter=32; kernel=(3x3); stride=(1x1) and padding=(1x1)
- FC1: Neurons=512; Dropout=0.2
- FC2: Neurons=256; Dropout=0.2
- FC3: Neurons=128
- # Parameters = 760,762
- Optimizer = Adam
- Learning Rate = 0.0001
- Weight Decay = 0.003

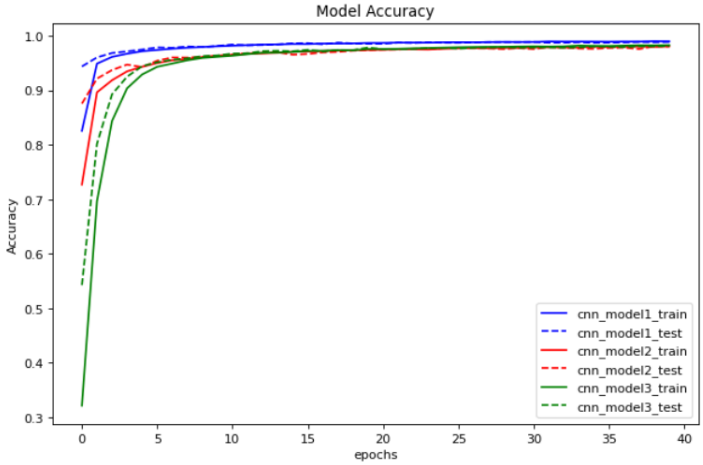
CNN 3:

- Convolution layer1: filter=8; kernel=(4x3x34); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- Convolution layer2: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- FC1: Neurons=120; Dropout=0.2
- FC2: Neurons=80; Dropout=0.2
- FC3: Neurons=50
- FC4: Neurons=24
- # Parameters = 110,652
- Optimizer = Adam
- Learning Rate = 0.0001
- Weight Decay = 0.003

Below is the plot of the training loss and the test loss for all the 3 models against the epochs



The plot below shows the model accuracy during training and validation versus the number of epochs



Conclusion:

It was observed that all three models had a good performance and fit well when tested on the validation dataset. However, comparing the 3 models from the above results, the first model (Model 1) with only one fully connected layer has the best model performance. The first model had the highest training and test accuracy and the lowest loss with the highest convergence.

Below is the training accuracy and training loss of the models

- Model1: Train Acc: 98.95% Train Loss: 0.03844
- Model2: Train Acc: 98.05% Train Loss: 0.06750
- Model3: Train Acc: 98.26% Train Loss: 0.06182

Below is the test accuracy and test loss of the models

- Model1: Test Acc: 98.85% Test Loss: 0.04050
- Model2: Test Acc: 98.01% Test Loss: 0.06590
- Model3: Test Acc: 98.11% Test Loss: 0.06462

PART 2: Optimization

2.1 Visualize Optimization Process

For this part, I trained a DNN model with the MNIST dataset, and the weights of the model were collected during training and plotted using PCA.

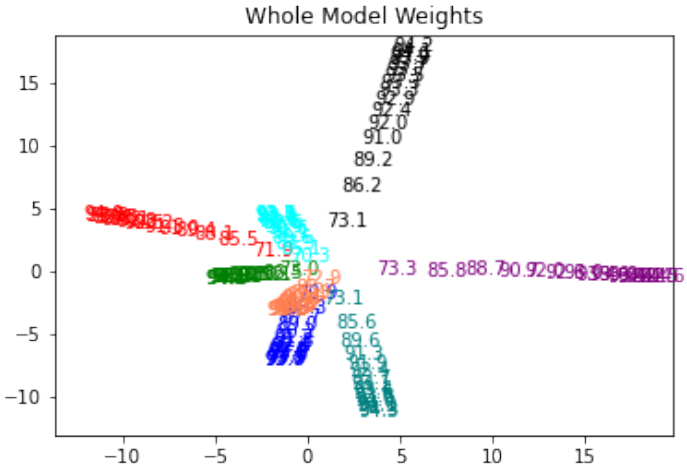
Model Architecture:

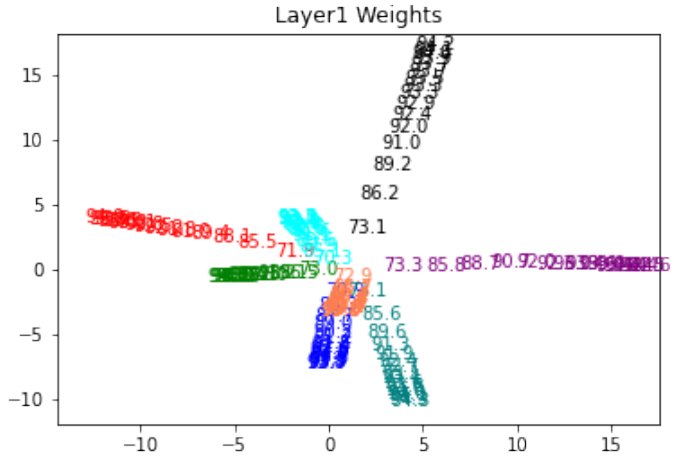
- 2 hidden layers
- Layer1: Neurons = 50; Activation Function= ReLU
- Layer2: Neurons = 30; Activation Function= ReLU
- Output logits: 10; Activation Function= SoftMax
- # Parameters = 41,090
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 0.001
- Epochs: 45

Eight iterations were performed during training, and for each iteration, the model was trained for 45 epochs. During training, the weights of the entire model were collected after every 3 epochs. The model of the first layer was also collected after every 3 epochs.

Principal Component Analysis (PCA) was applied to the collected weights to reduce the dimensions to 2.

The plot below shows the PCA plot for the whole model with respect to the model's accuracy after every 3 epochs.





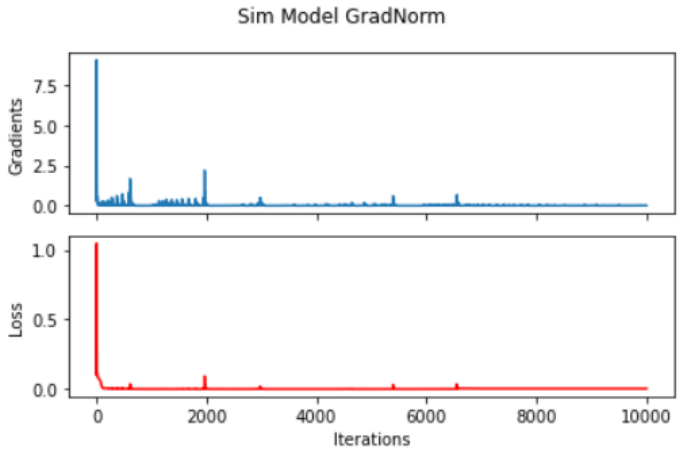
2.2 Observe Gradient Norm During Training

For this task, the gradient norm and loss of the model were recorded during training at every iteration. This task was carried out on both the simulated sin function and the MNIST dataset.

- **Simulated Function:** $\frac{\sin(5\pi x)}{5\pi x}$

The sin function was trained on a DNN model with the architecture shown below:

- Layer1: Neurons = 170; Activation Funct= ReLU
- Layer2: Neurons = 200; Activation Funct= ReLU
- Output logits: 1
- # Parameters = 34,741
- Loss Function: nn.MSELoss()
- Optimizer = Adam
- Learning Rate = 0.01
- Epochs: 10,000



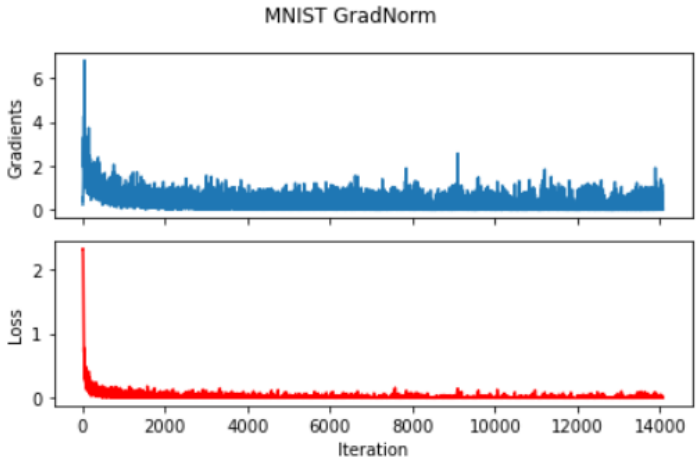
It was observed that for both models, the gradient norm decreases as the model loss decreases. There was a sharp increase in the gradient norm at about 2000

epochs, but from the loss plot, we can see that there was also an increase in the loss value at the same iteration

- **MNIST Dataset Grad Norm**

The mnist dataset was trained on a CNN function and below is the model architecture used:

- Convolution layer1: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- Convolution layer2: filter=32; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- FC1: Neurons=512
- FC2: Neurons=256
- Output logits: 10; Activation Funct= SoftMax
- # Parameters = 942,026
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 0.001
- Epochs: 40



Conclusion:
It was observed that for both models, the gradient norm decreases as the model loss decreases.

2.3 What happened when Gradient is Almost Zero
The simulated sin function was trained for 300 epochs. The gradient norm equal to zero was computed by changing the objective loss function to gradient norm. The Hessian matrix of the loss of grad norm was calculated to compute the minimal ratio. The proportion of the eigenvalues greater than zero was defined as the minimal ratio.

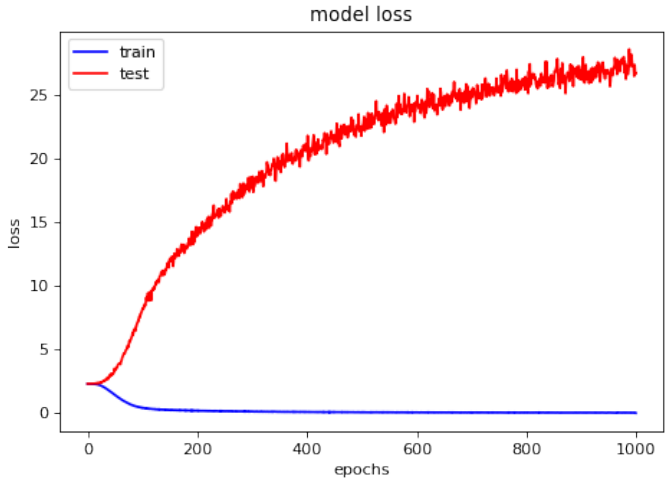
PART 3: Generalization

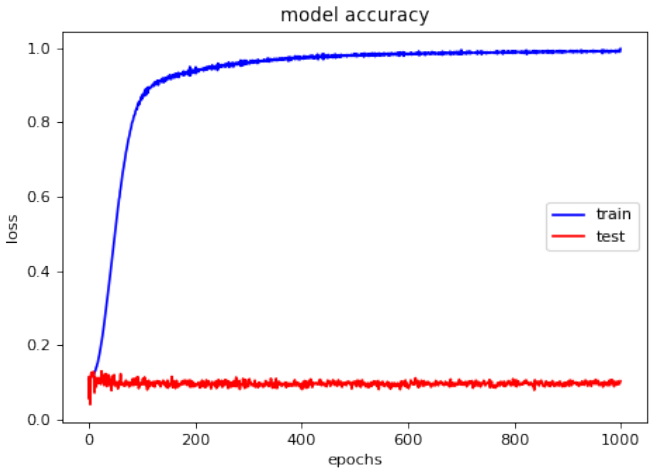
3.1 Can Model Fit Random Labels?
For this task, the MNIST dataset was trained with a DNN of 3 hidden layers with 1,600 hidden units across the 3 layers. The labels of the training dataset were shuffled before training.

Model Structure:

- 3 hidden layers
- Layer1: Neurons = 1024; Activation Funct.= ReLU
- Layer2: Neurons = 512; Activation Funct.= ReLU
- Layer3: Neurons = 64; Activation Function= ReLU
- Output logits: 10; Activation Function= SoftMax
- # Parameters = 1,362,122
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 0.0001
- Epochs: 1000
- Training batch size = 64

The plot below shows the training and test loss





Conclusion:
The training loss was observed to reduce after every epoch until it reached convergence and the training accuracy also increased. This shows that the model can fit random labels during training and achieve high performance during training. However, the model could not fit the test data and performed poorly during the test. Test loss increased after every epoch, and the test accuracy decreased.

3.2 Number of parameters v.s. Generalization

For this task, 13 CNN models were trained with different number of parameters for each model. Each model had the same number of fully connected layers (3) but each layer for these models had different number of hidden units. Hidden units for each layer of the models were stored in a list and shown below:

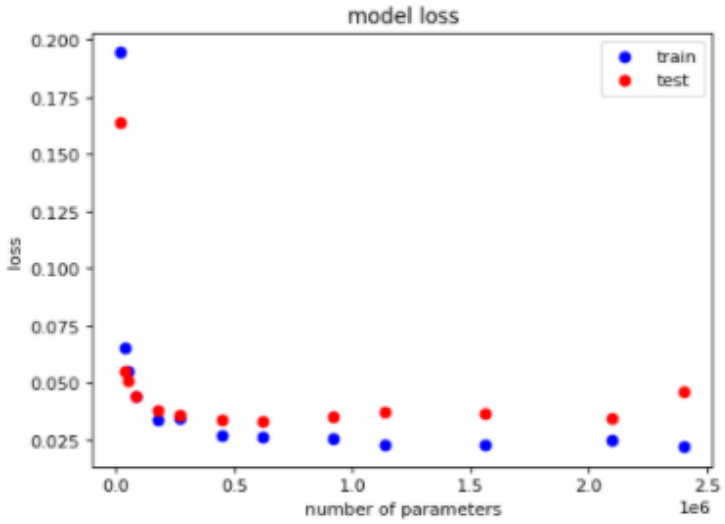
Layer1 = [10,20,30,50,100,150,250,350,500,600,800,1000,1024]
Layer2 = [5,10,35,50,100,128,150,150,200,250,300,400,512]
Layer3 = [5,20,20,25,50,64,70,100,128,150,200,300,512]

The 13 models had the following total number of parameters respectively: 20635, 36820, 53885, 87335, 177360, 268384, 445980, 622710, 916518, 1135610, 1562510, 2097510 and 2404042

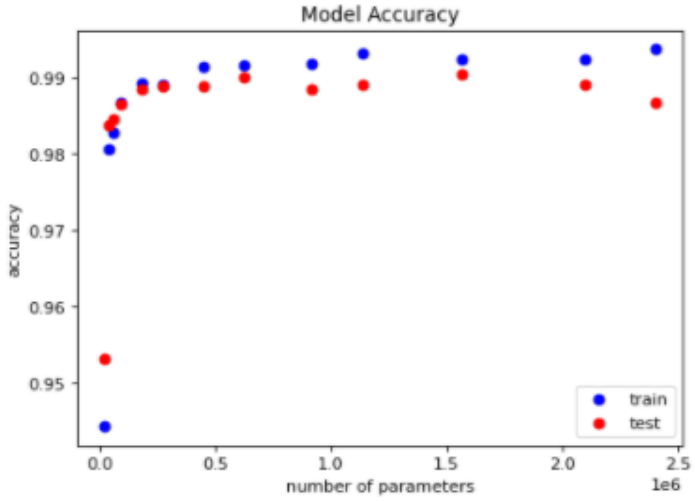
Model Structure:

- Convolution layer1: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- Convolution layer2: filter=32; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- 3 fully connected layers
- Activation Function: ReLU
- Output logits: 10; Activation Funct= SoftMax
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 0.001
- Epochs: 5

The plot below shows the training and test loss of each model in comparison with the number of parameters



The plot below shows the training and test accuracy of each model in comparison with the number of parameters



Conclusion:
As the number of model parameters increases, the model's accuracy increases, and the model's loss decreases. However, the test loss begins to increase while the test accuracy increases after a while. This is because as the number of parameters increases, the model tends to overfit during training, which will cause the model not to fit well with the test data.

3.3.1 Flatness v.s. Generalization – part 1

In this section, four CNN models were trained with different training approaches. Two of the models were trained with two different batch sizes, and the other two models were trained with different learning rates. The same model structure was used with the structure shown below;

Model Structure:

- Convolution layer1: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- Convolution layer2: filter=32; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- FC1: Neurons=200; Activation Function= ReLU
- FC2: Neurons=100 ; Activation Function= ReLU
- FC2: Neurons=14; Activation Function= ReLU
- Output logits: 10; Activation Funct= SoftMax
- # Parameters = 340264
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 0.001

Models with Batch size: 64 and Batch Size: 1024

Two models were trained with batch sizes of 64 and 1024, respectively. Each model was trained for 5 epochs, and the parameters of each model were collected after training.

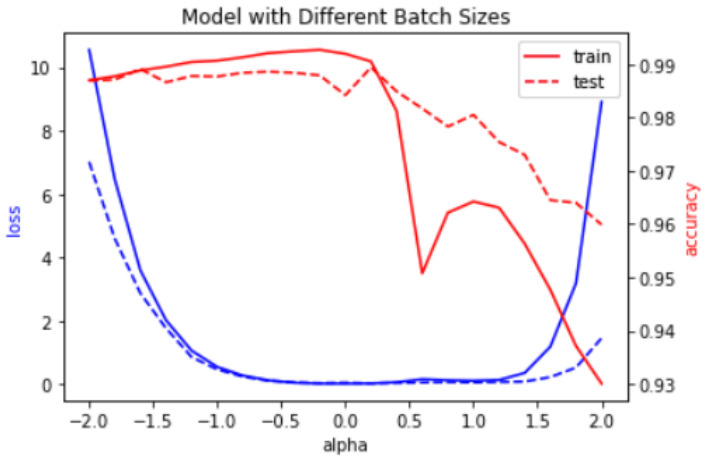
The interpolation of the parameters of the two models was calculated using the formula below;

$$\theta_\alpha = (1 - \alpha)\theta_1 + \theta_2$$

Alpha (α) is the interpolation ratio, θ1 is the model parameter with 64 batch sizes, and θ2 is the second model parameter with 1024 batch sizes. 21 different alpha values between the range of -2 and 2.0 were generated to compute the linear interpolated parameters between the two model parameters.

Hence, 31 different interpolated parameters were generated, and each parameter value was vectorized and passed to new models for training. A total of 31 new models with different interpolated parameters based on alpha values were trained with a batch size of 128. The training loss and train accuracy of each model was computed. Each model was also tested with a batch size of 128 on the test dataset.

The graph below is the plot of the loss and accuracy for the model with different alpha values



Conclusion

The plot above illustrates that the model performance increases in accuracy and decreases in loss as the alpha value increases from -2 to 0.0. However, the model's accuracy decreased when alpha was between 0.5 and 0.8, slightly increasing between 0.8 and 1.3 of alpha. Finally, the model's performance begins to plummet with an increase in alpha from 1.3 upwards.

Learning rate: 1e-2 and Learning Rate: 1e-3

Two models were trained with different learning rates:

- Learning Rate 1 = 1e-2
- Learning Rate 2 = 1e-3

Each model was trained for 5 epochs with batch_size=128, and the parameters of each model were collected after training.

The interpolation of the parameters of the two models was calculated using the formula below;

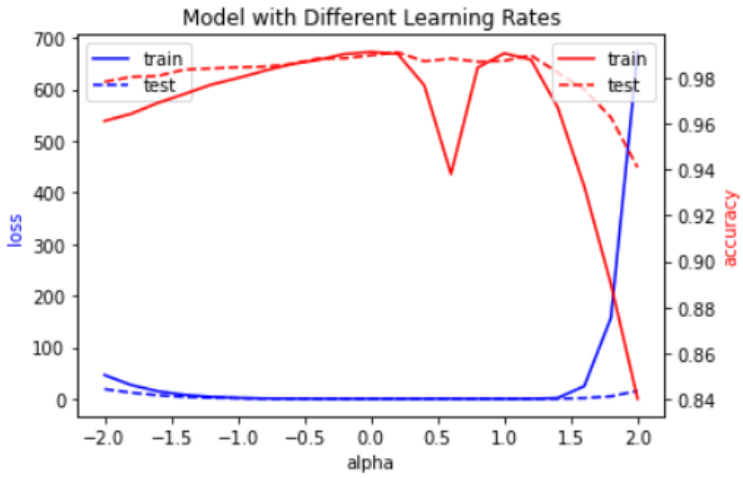
$$\theta_\alpha = (1 - \alpha)\theta_1 + \theta_2$$

Alpha (α) is the interpolation ratio, θ1 is the model parameter with a learning rate=1e-2, and θ2 is the second model parameter with a learning rate=1e-3. 21 different alpha values between the range of -2 and 2.0 were generated to compute the linear interpolated parameters between the two model parameters.

Hence, 31 different interpolated parameters were generated, and each parameter value was vectorized and passed to new models for training. A total of 31 new models with different interpolated parameters based on alpha values were trained with a batch size of 128 and a learning rate of 0.001. The training loss and train accuracy of each model was computed. Each

model was also tested with a batch size of 128 on the test dataset.

The graph below is the plot of the loss and accuracy for the model with different alpha values



Conclusion

The plot above was generated with models with learning rates. Similar to the plots of the batch size, it was observed that the model had good performance with an increase in training and test accuracy and a decrease in both train and test loss as the alpha value increases from -2 to 0.0. However, when alpha was between 0.4 and 0.6, the model's training accuracy decreased to 0.94 and slightly decreased test accuracy. There was an increase in the model's accuracy between 0.8 and 1.2. Finally, when the alpha value increased from 1.6 upwards, the model's performance began to plummet with a high decrease in train and test accuracy and a high increase in train and test loss.

3.3.2 Flatness v.s. Generalization - Part2 (Sensitivity)

For this task, I trained 14 CNN models with the same number of parameters but varying batch sizes to load the training dataset unto the model during training. Below is the model architecture used across the 14 models

Model Structure:

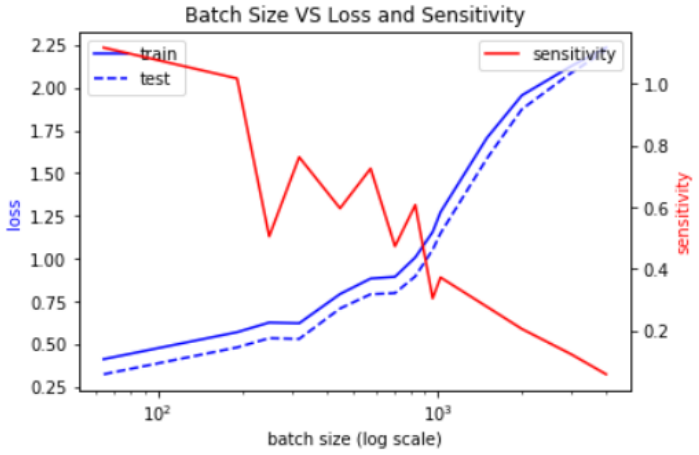
- Convolution layer1: filter=16; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- Convolution layer2: filter=32; kernel=(3x3); stride=(1x1) and padding=(1x1)
- Max pooling: kernel=(2x2); stride=(2x2)
- FC1: Neurons=1000; Activation Function= ReLU
- FC2: Neurons=500 ; Activation Function= ReLU
- FC2: Neurons=124; Activation Function= ReLU
- Output logits: 10; Activation Funct= SoftMax
- # Parameters = 2,137,674
- Loss Function: nn.NLLLoss()
- Optimizer = Adam
- Learning Rate = 1e-5

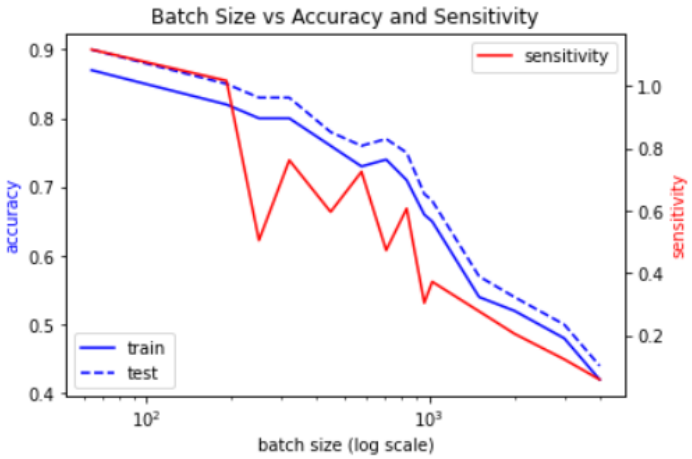
The list of batch sizes below was used for each model respectively.

batch_sizes = [64, 192, 250, 320, 448, 576, 704, 832, 960, 1024, 1500, 2000, 3000, 4000]

During training, the sensitivity of each model was calculated and recorded by computing the Frobenius norm of gradients of loss to input before the optimizer step to update weights to allow us to observe how sensitive the neural network model is to varying batch sizes.

The plot below shows the relationship between the model train and test loss, sensitivity, and batch sizes





Conclusion:

The above results show that the model train and test performance decreases as the batch size increases. Model loss increases, and the accuracy decreases as batch size increases. However, the sensitivity of the model decreases as batch size increases. Therefore, the best model performance is achieved with a small batch size, and the model can generalize better with unseen data.