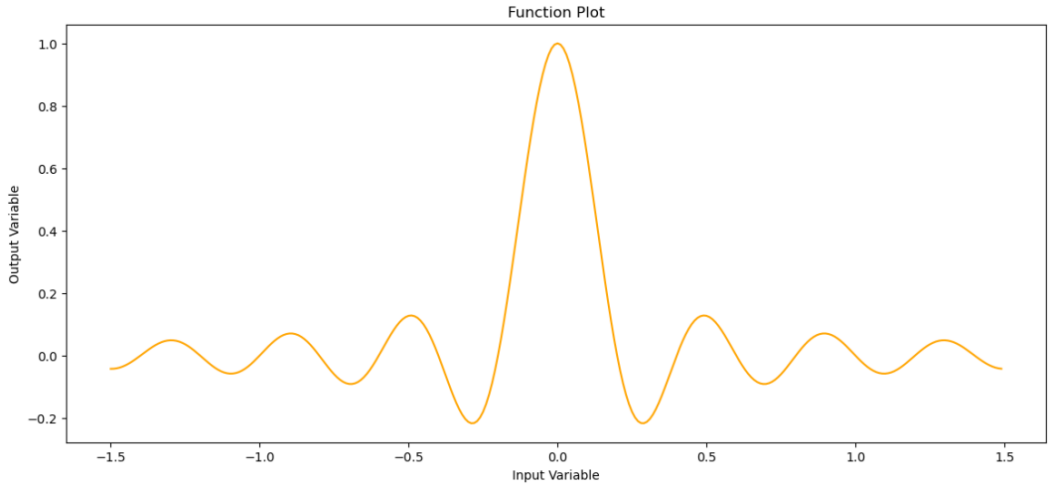
THARUN KUMAR VASPARI -CPSC-8430-DEEP-LEARNING-HW1

HW 1-1 Deep vs Shallow

1-1-1. Simulate a Function

https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_SIMULATE_FUNCTION.ipynb

Function 1: sin(5*pi*x) / 5*pi*x
The graph of this expression is displayed below.



Simulation of a Function:

Models: Three distinct models were created. To regularize these models, a weight decay parameter was included, which adds a penalty to the neural network's cost function. This change resulted in a decrease in weight values during the back-propagation process.

MODEL 1:

- Number of Dense Layers: 7
- Number of Parameters: 571
- Activation Function: Leaky ReLU
- Optimizer: RMSprop
- Learning Rate: 1×10^{-3}

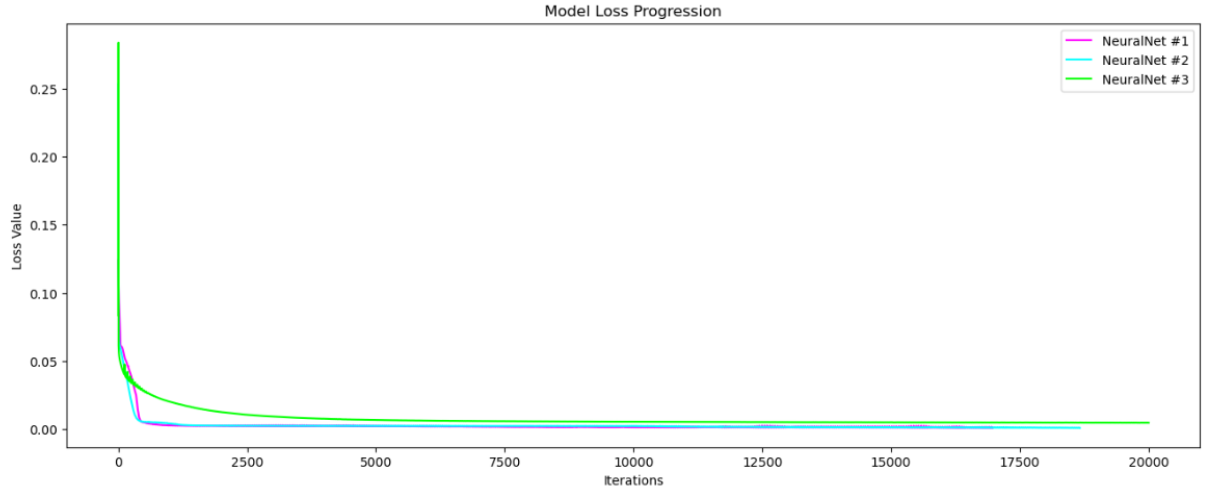
MODEL 2:

- Number of Dense Layers: 4
- Number of Parameters: 546
- Activation Function: Leaky ReLU
- Optimizer: RMSprop
- Learning Rate: 1×10^{-3}

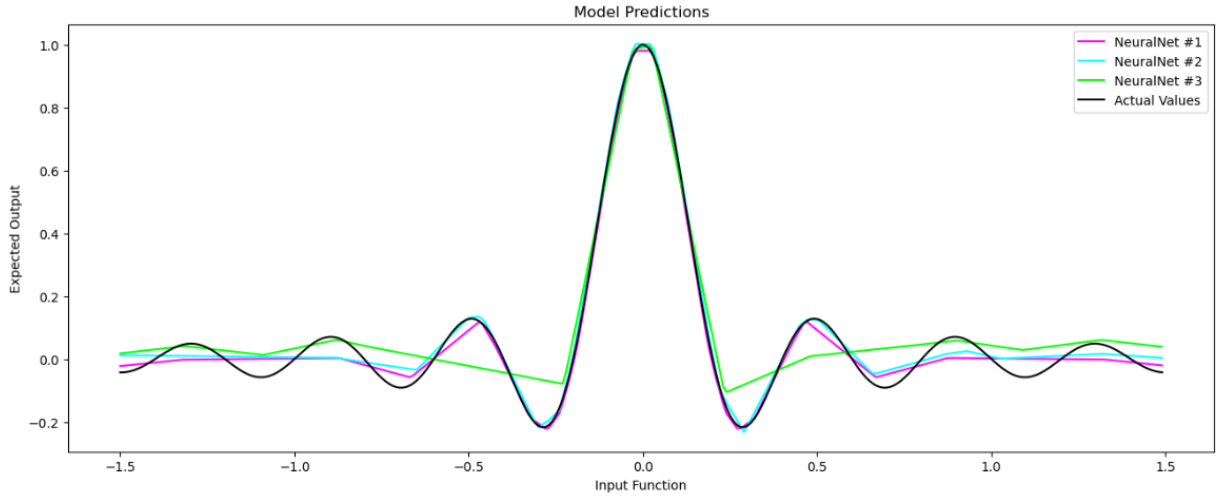
MODEL 3:

- Number of Dense Layers: 2
- Number of Parameters: 546
- Activation Function: Leaky ReLU
- Optimizer: RMSprop
- Learning Rate: 1×10^{-3}

The models achieved convergence when the learning rate sharply decreased or when they reached the maximum number of epochs. The loss for every model is shown in the graph below:



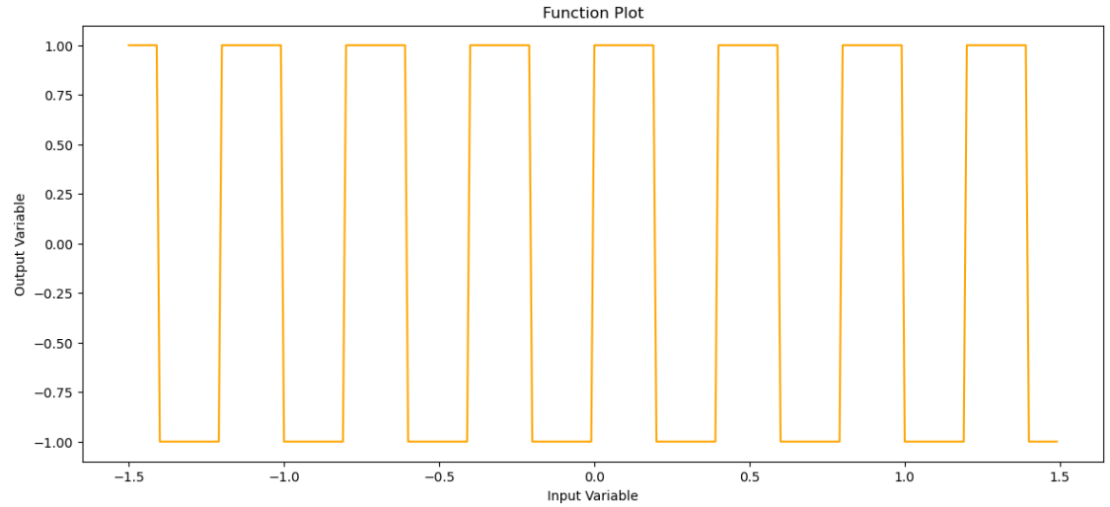
A comparison between the true values and the predictions generated by the three models. is indicated in the graph below.



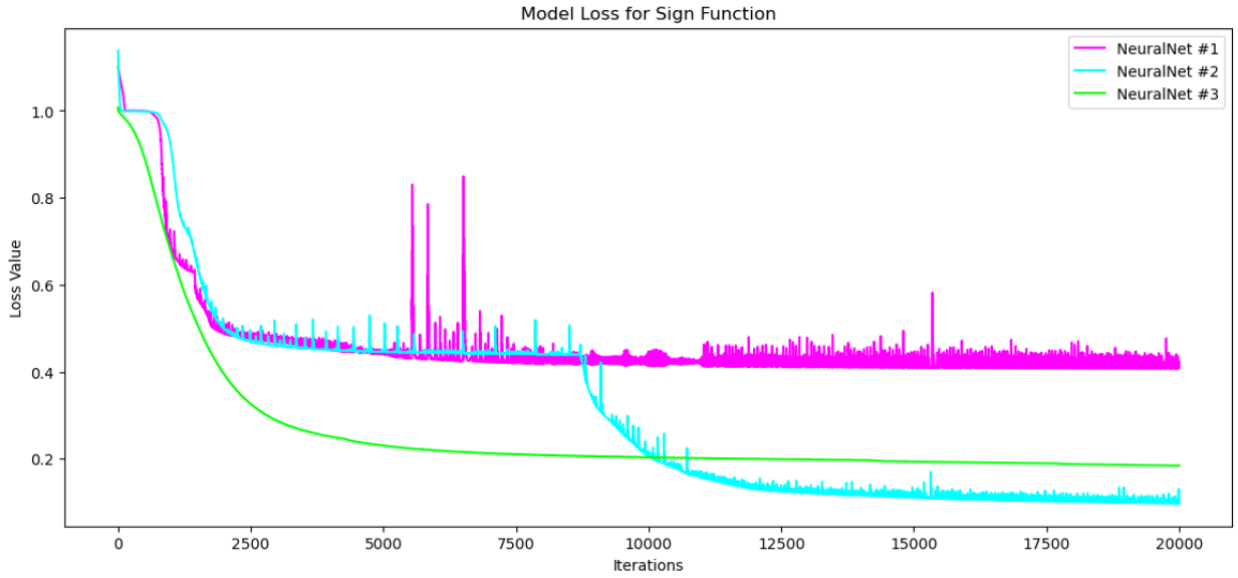
Review:

Models 1 and 2 reached convergence quickly, while Model 3 required the most epochs to converge. The graph shows that Models 1 and 2 had lower loss values. In summary, models with more dense layers learned faster and more efficiently.

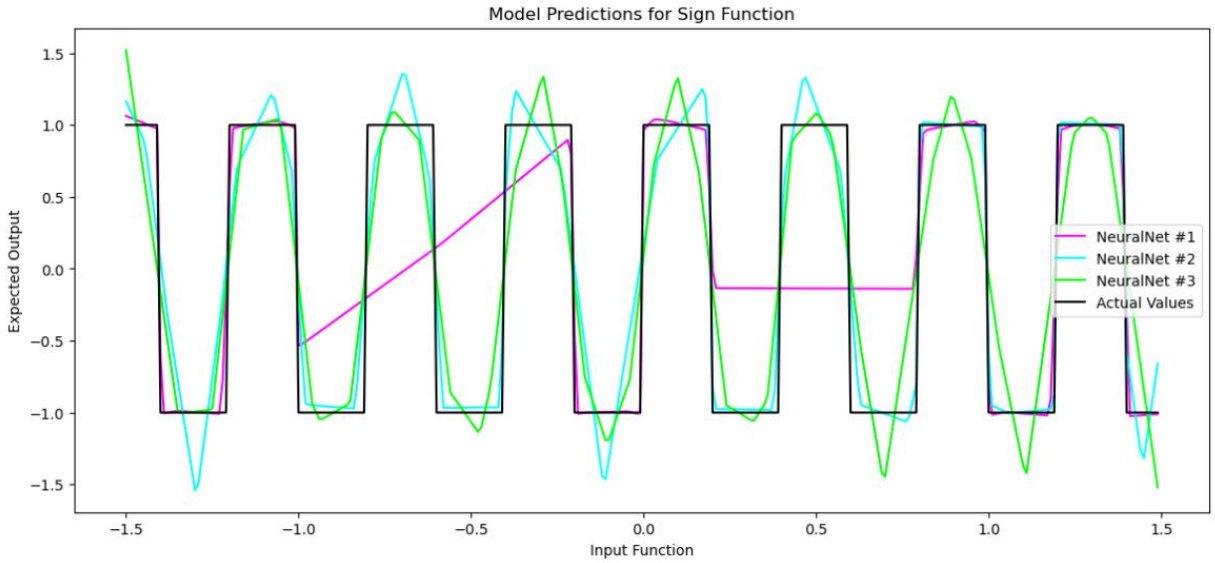
Function 2: sgn(sin(5*pi*x) / 5*pi*x)
The graph of this expression is displayed below.



After reaching the maximum training iterations, all models converged. The graph below illustrates the loss for each model.



The contrast between the actual and anticipated values for the models discussed is shown in the graph below.



Review:

All the models reached the maximum number of epochs before achieving full convergence. But in the end, Model 1 produced the lowest loss and converged with the real values, marginally exceeding Model 2. Conversely, Model 3 did not converge and was unable to obtain a low loss.

1-1-2. Train on Actual Task:

https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_Train_On_Actual_Task_MNIST.ipynb

Model 1: LeNet (Custom LeNet)

- Conv Layer 1: 1 input → 6 output channels with 5x5 kernels.
- Conv Layer 2: 6 input channels → 16 output channels with 5x5 kernels.
- Dense Layer 1: 16 feature maps of 5x5 size are flattened to form a vector, then passed through a layer with 120 neurons.
- Dense Layer 2: 120 neurons → 84 neurons.
- Dense Layer 3: 84 neurons → 10 outputs.
- Activation Function: ReLU
- Pooling: Max Pooling

Model 2: CNN (CustomCNN)

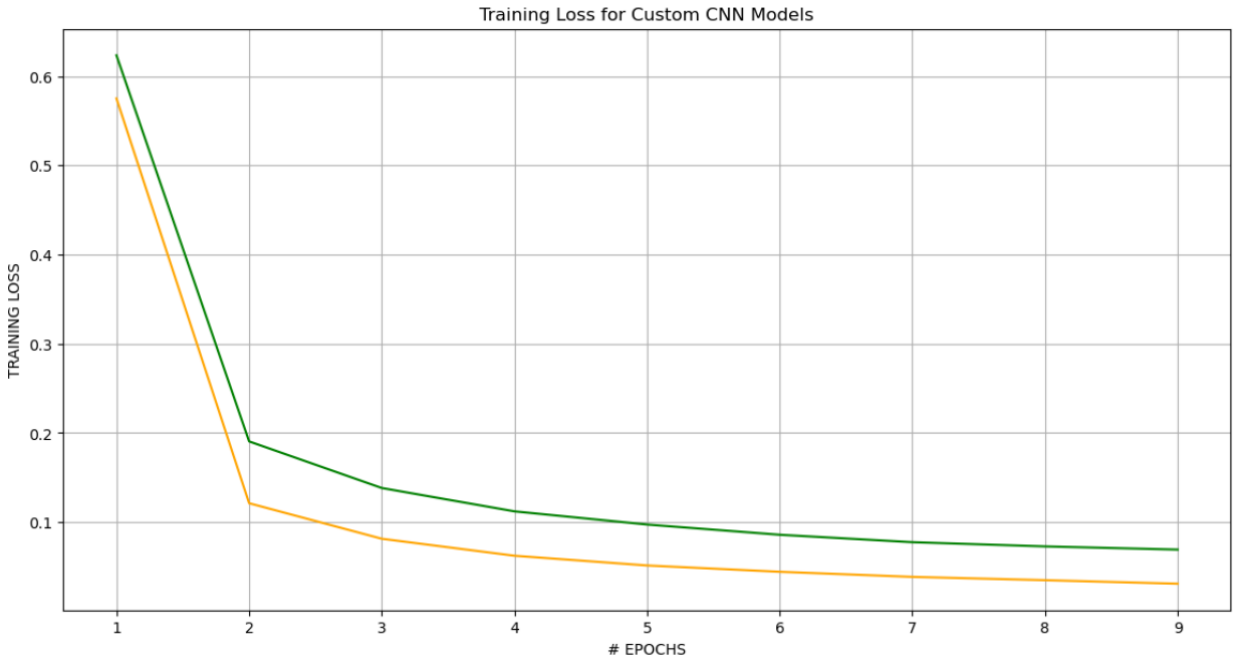
- Conv Layer 1: 1 input channel → 32 output channels with 5x5 kernels.
- Conv Layer 2: 32 input channels → 32 output channels with 5x5 kernels.
- Conv Layer 3: 32 input channels → 64 output channels with 5x5 kernels.

- Fully Connected Layer 1: The flattened output from the convolutional layers is fed into a 256-unit fully connected layer.
- Fully Connected Layer 2: 256 units → 10 units.
- Dropout: Dropout (0.5) is used after some layers to prevent overfitting.
- Pooling: Max Pooling
- Activation Function: ReLU

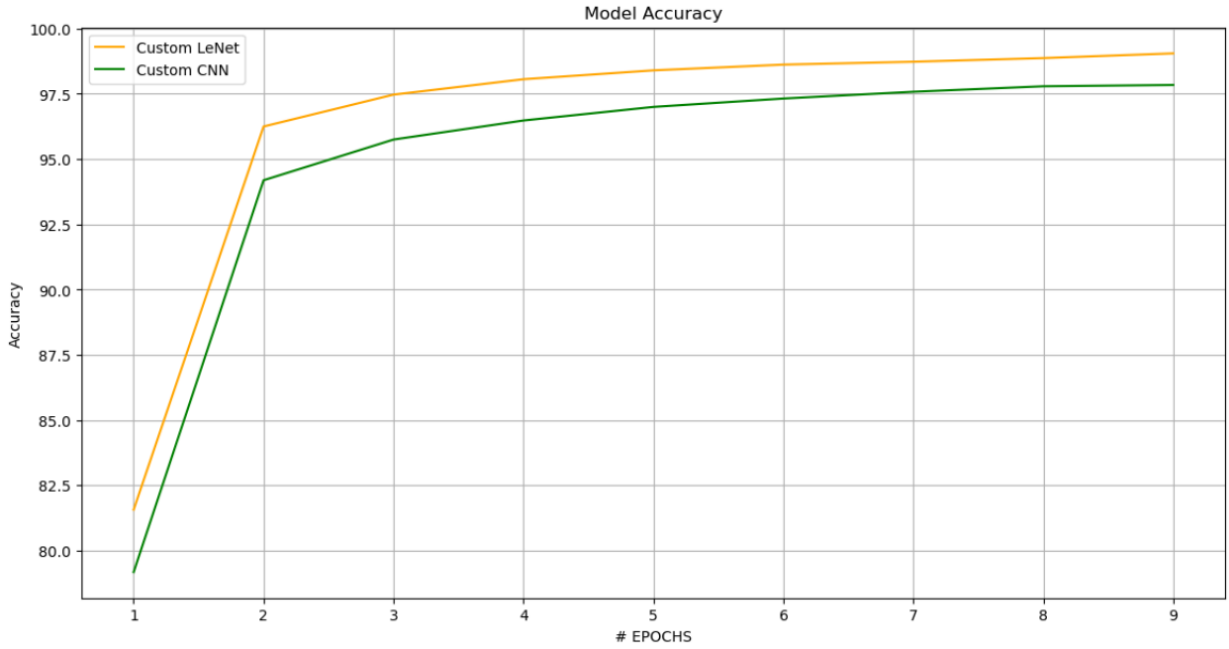
Parameters:

- Learning rate (lr): 0.01
- Loss function: Cross-Entropy Loss (torch.nn.CrossEntropyLoss())
- Batch size: training is 64, testing is 1000
- Number of epochs: 10
- Optimizer: Stochastic Gradient Descent (SGD)

The following image represents the training loss for **CustomLeNet** and **CustomCNN**.



The image below shows the training accuracy for **CustomLeNet** and **CustomCNN**.



Review:

CustomLeNet showed less training loss and outperformed CustomCNN. The architecture of CustomLeNet, an optimized CNN based on the LeNet structure and surpassed the performance of the custom-designed CNN.

HW 1-2 Optimization

1-2-1. Visualize the Optimization Process.

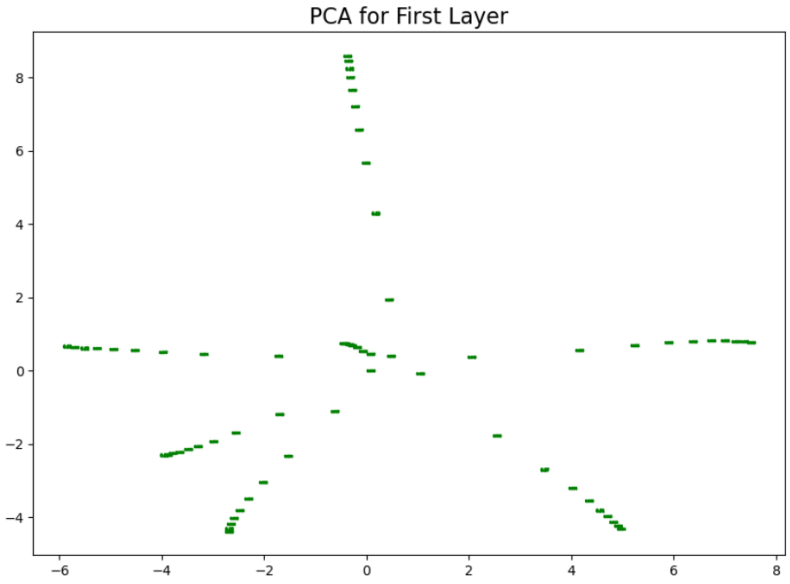
https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_Principal_Component.ipynb

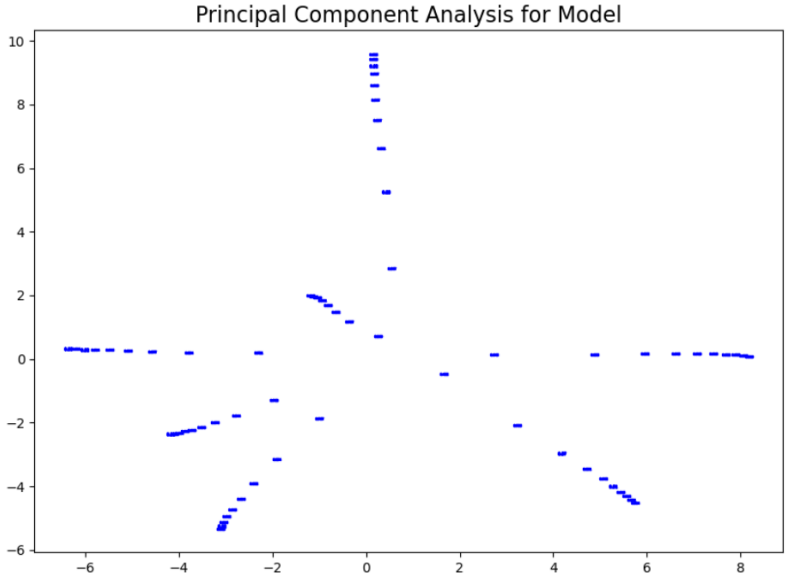
The following model has been trained using the MNIST dataset.

Experiment Settings:

- Input Layer: Dense layer with 784 input units.
- Hidden Layer 1: Fully Connected Layer with 10 units.
- Hidden Layer 2: Dense Layer with 20 units.
- Output Layer: Dense layer with 10 output neurons.
- Number of Epochs: 29
- Batch Size: 1000 for both training and testing datasets.
- Loss function: Cross Entropy Loss function
- Dimension Reduction Method: Principal Component Analysis (PCA)
- Optimizer: Adam optimizer

- Learning rate: 0.0004



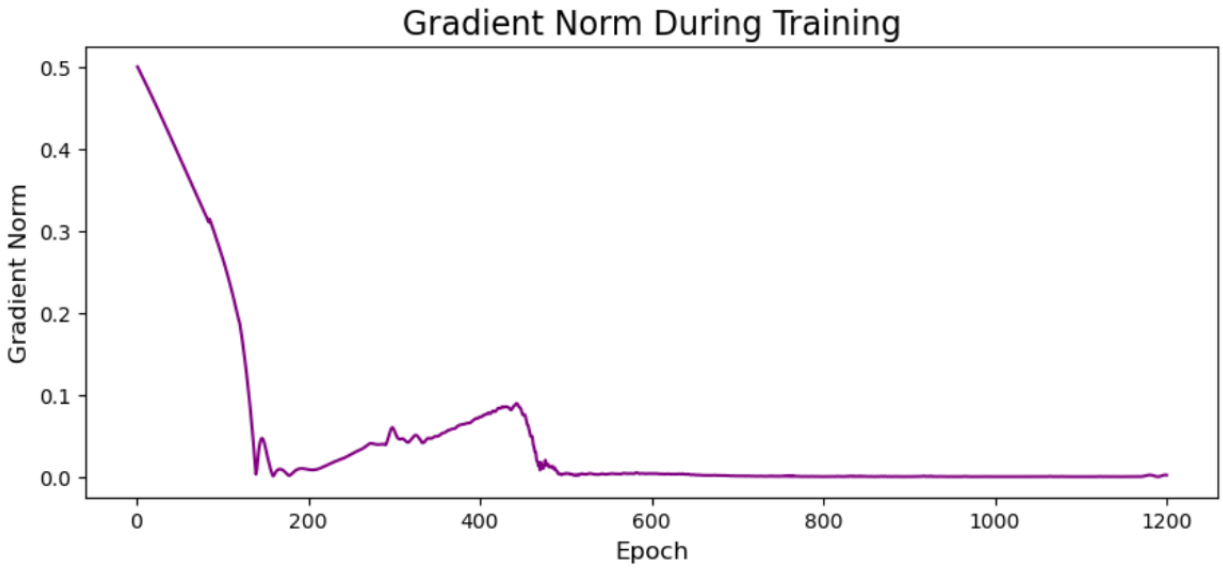


1-2-2. Observe gradient norm during training

https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_gradnorm.ipynb

sin(5*pi*x) / 5*pi*x function is used to calculate the Gradient norm and loss.

The gradient norm during training is shown in the graph below.



The training loss graph throughout the epochs is shown below.

Review:

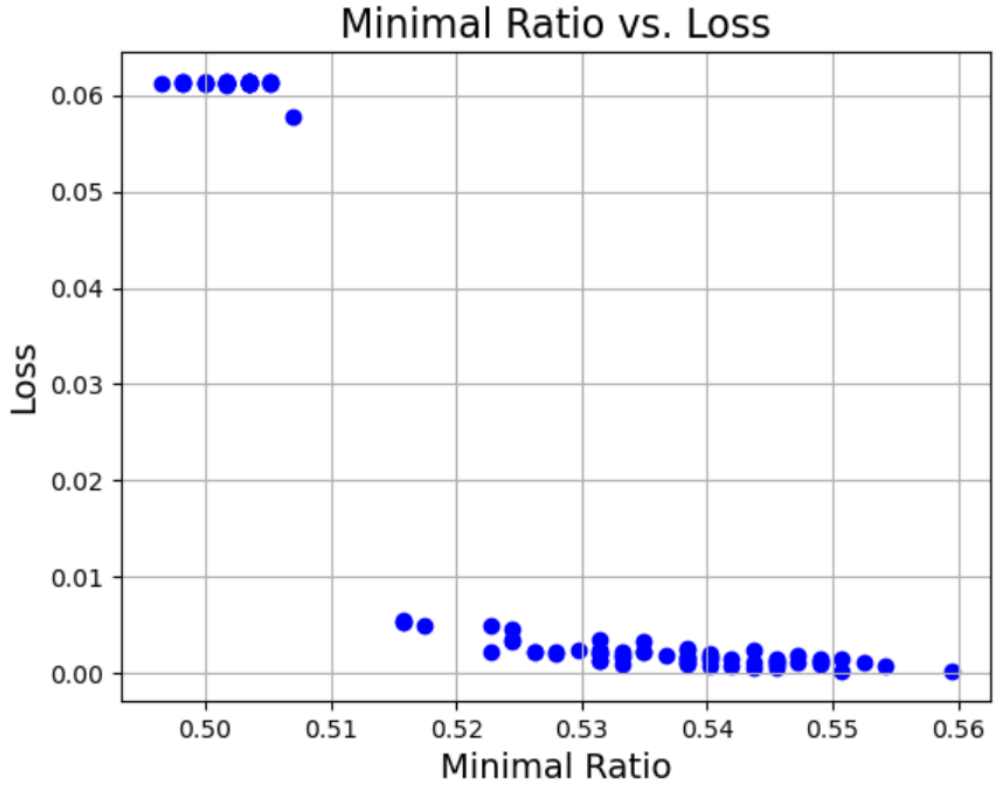
Throughout the training process, the gradient norm has shown a steady decline, indicating that the model is stabilizing. Concurrently, the training error has also been decreasing, reflecting improved performance as the training iterations reach their maximum limit. This trend suggests that the model is effectively learning from the data and making significant progress over time.

1-2-3. What happens when gradient is almost zero?

https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/min_grad.ipynb

When the gradient norm drops below a predetermined threshold (e.g., 0.001), the model parameters have converged, and future updates are probably going to be small. This weight at

which the gradient norm is zero is found during training. A larger minimal ratio indicates that the model is in a stable area of the loss landscape. The minimal ratio is defined as the fraction of positive eigenvalues of the Hessian matrix, which is derived using the second-order derivatives of the loss function.



The above model was trained 100 times, and the minimal ratio, representing the proportion of positive eigenvalues of the Hessian matrix, was recorded alongside the corresponding loss values. The resulting plot illustrates the relationship between the minimal ratio and loss, providing insights into the stability of the model's training process.

Review:

The minimal ratio against loss plot shows that greater minimal ratios are typically correlated with lower loss values, indicating that the model is successfully locating stable minima in the loss landscape. To achieve maximum performance during training, it is important to strike a balance between loss reduction and model stability. In certain circumstances, increased loss values accompanied by high minimal ratios may indicate overfitting or instability.

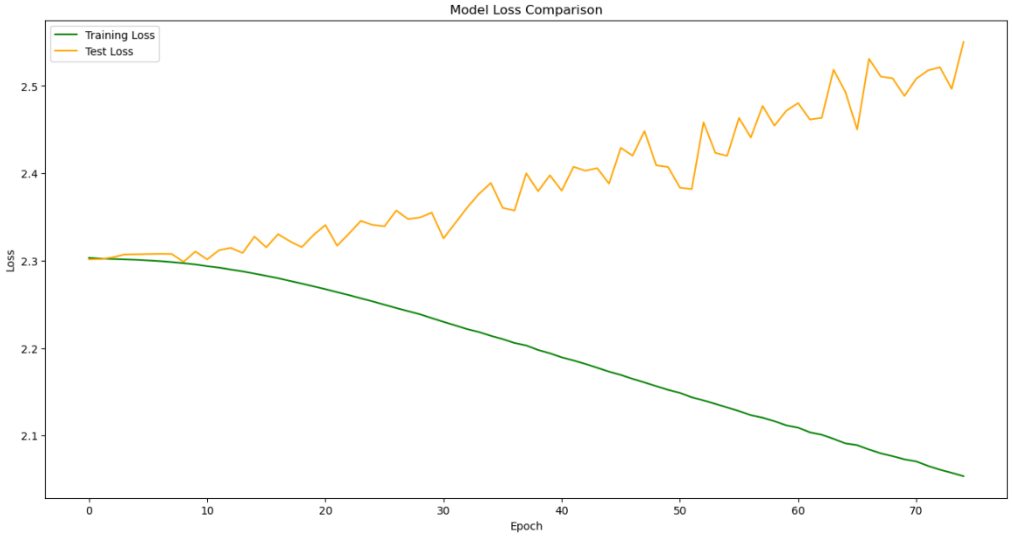
1-3. Generalization

1-3-1. Can network fit random labels?

https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_Labelfit.ipynb

Model 1: LeNet (Custom LeNet)

- Conv Layer 1: 1 input channels → 6 output channels with 5x5 kernels.
- Conv Layer 2: 6 input channels → 16 output channels with 5x5 kernels.
- Dense Layer 1: 16 feature maps of 5x5 size are flattened to form a vector, then passed through a layer with 120 neurons.
- Dense Layer 2: 120 neurons → 84 neurons.
- Dense Layer 3: 84 neurons → 10 outputs.
- Activation Function: ReLU



Review:

The model is trained on randomly assigned labels, which slows down the training process. As the model progresses through the epochs, it attempts to memorize these labels, resulting in a decrease in training loss. However, the test loss continues to rise over time.

1-3-2. Number of parameters vs Generalization

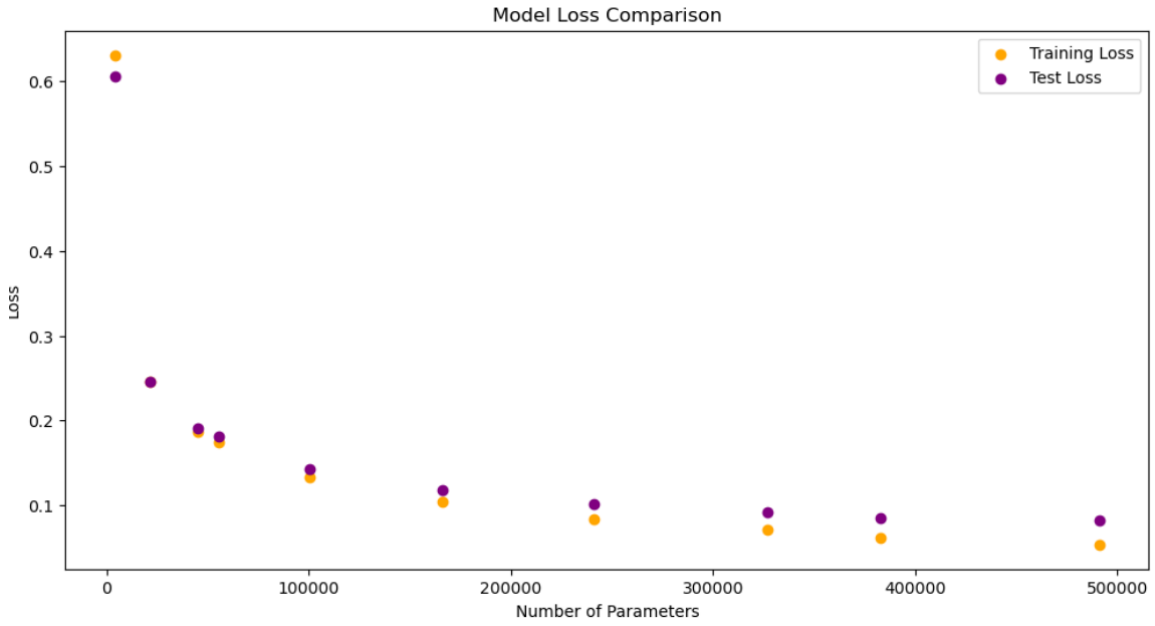
https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_PARAMETER_COMPARE.ipynb

Experiment Settings:

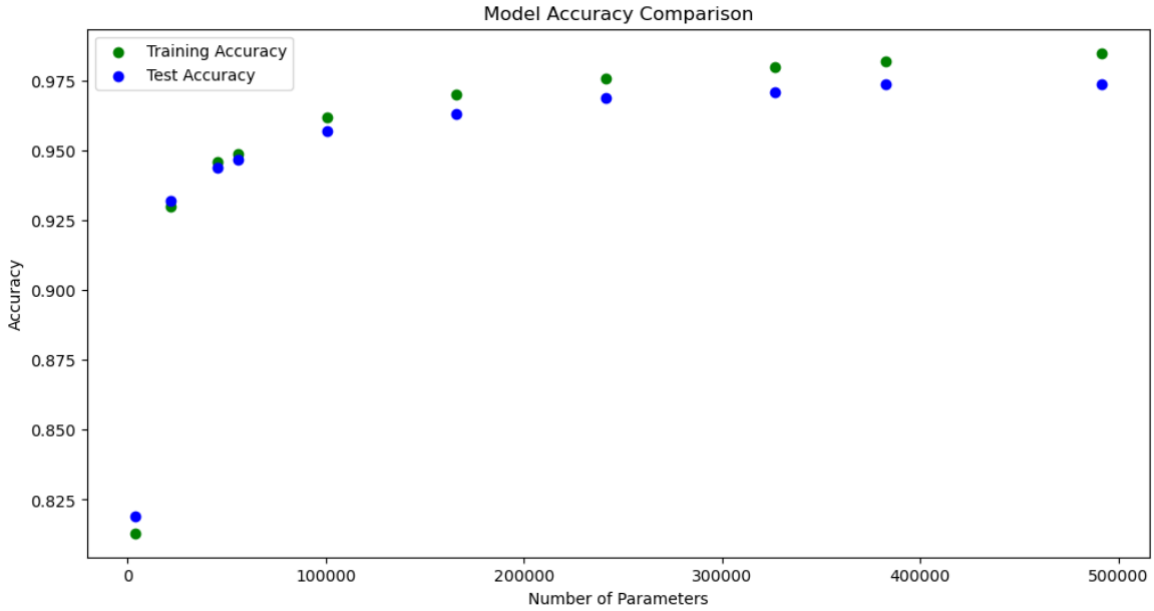
- Model 1: 3 layers (1 hidden layer with 5 neurons)
- Model 2: 3 layers (1 hidden layer with 25 neurons)
- Model 3: 3 layers (1 hidden layer with 50 neurons)
- Model 4: 3 layers (1 hidden layer with 60 neurons)
- Model 5: 3 layers (1 hidden layer with 100 neurons)
- Model 6: 3 layers (1 hidden layer with 150 neurons)
- Model 7: 3 layers (1 hidden layer with 200 neurons)
- Model 8: 3 layers (1 hidden layer with 250 neurons)
- Model 9: 3 layers (1 hidden layer with 280 neurons)

- Model 10: 3 layers (1 hidden layer with 350 neurons)
- Batch Size: Training 50 and Testing 100.
- Number of Epochs: 8
- Learning rate: 0.0001

The graph below compares the test loss and training loss across different models as function of the number of parameters.



The graph below illustrates the accuracy of different models during testing and training concerning the number of parameters.



Review:

As the number of parameters increases, the disparity between test loss and accuracy for both training and testing becomes widened. The test loss tends to stabilize much earlier than accuracy or training loss. This occurs because the model has more parameters to learn from, leading to overfitting.

1-3-3. Flatness vs Generalization

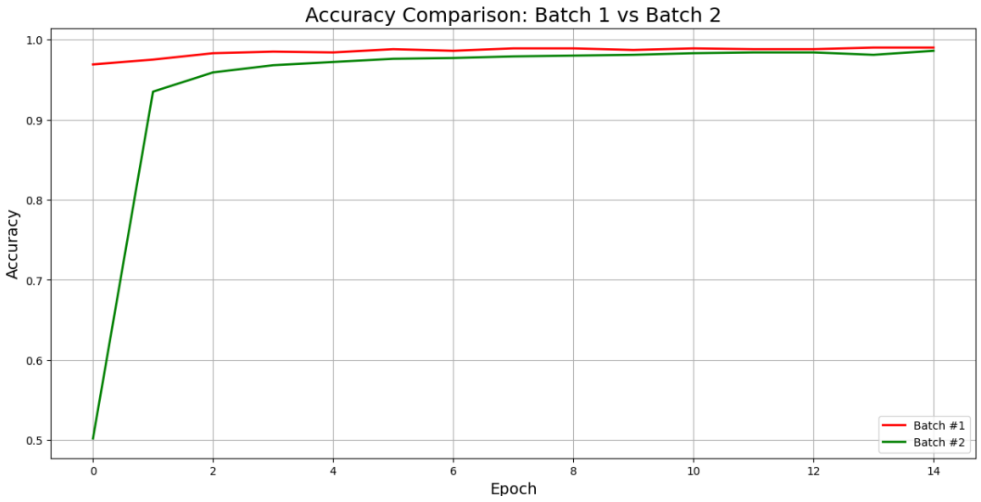
https://github.com/VASPARITHARUNKUMAR/CPSC-8430---Deep-Learning---HW1/blob/main/HW1_Interpolation.ipynb

The model presented below has been trained on the MNIST dataset.

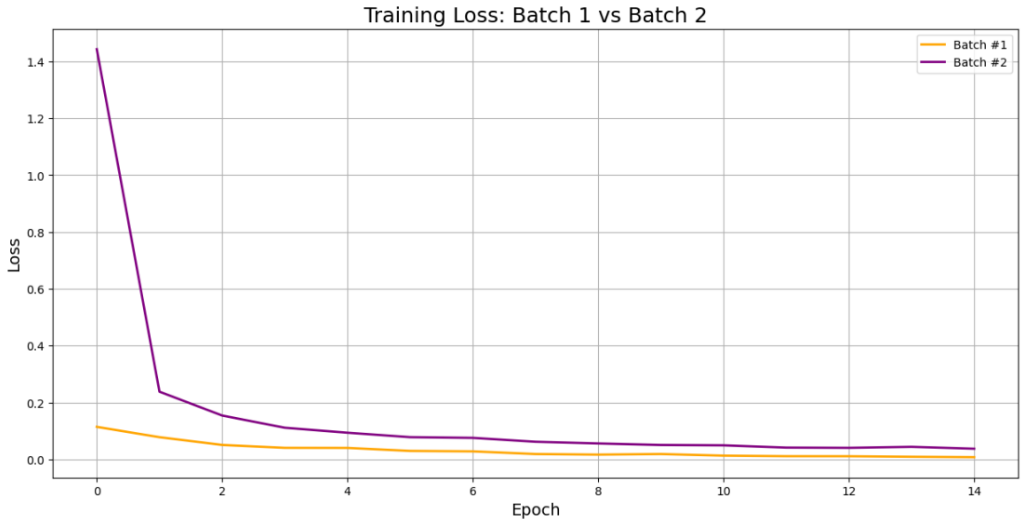
Experiment Settings:

- Epochs: 15
- Optimizer: Stochastic Gradient Descent (SGD)
- Convolutional Layers:
  - Layer 1: 1 input → 6 output channels, with 5*5 kernals.
  - Layer 2: 6 input channels → 16 output channels, with 5*5 kernals.
- Fully Connected Layers:
  - The first layer takes 16 * 5 * 5 (400) features as input and outputs 120, the second outputs 84, and the final outputs 10 classes (digits 0-9).

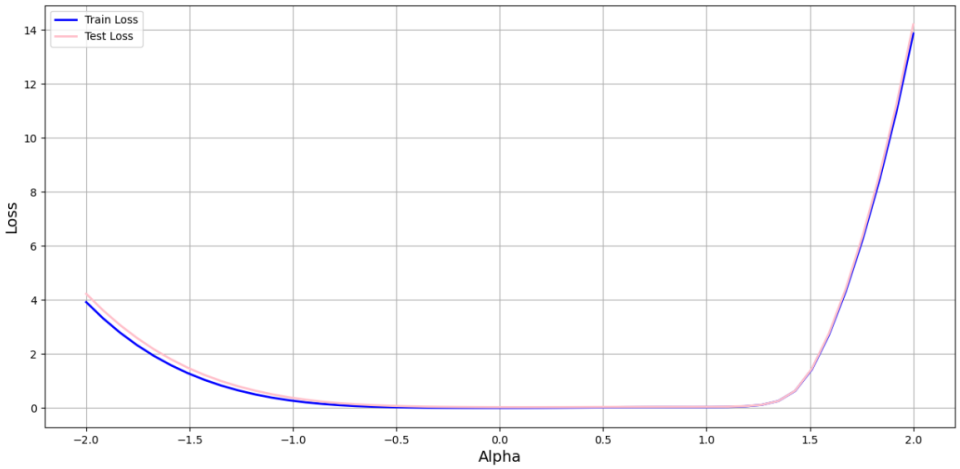
The graph below shows the comparison between testing accuracy and training accuracy with number of epochs.



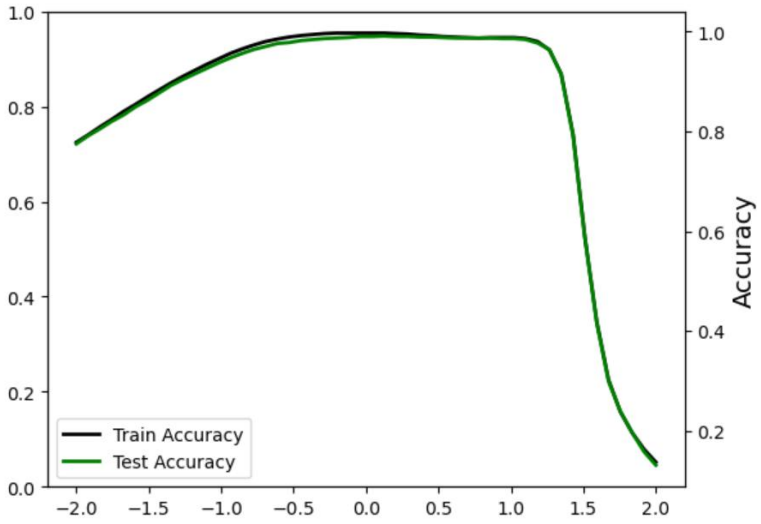
The graph below illustrates the comparison between testing and training with number of epochs.



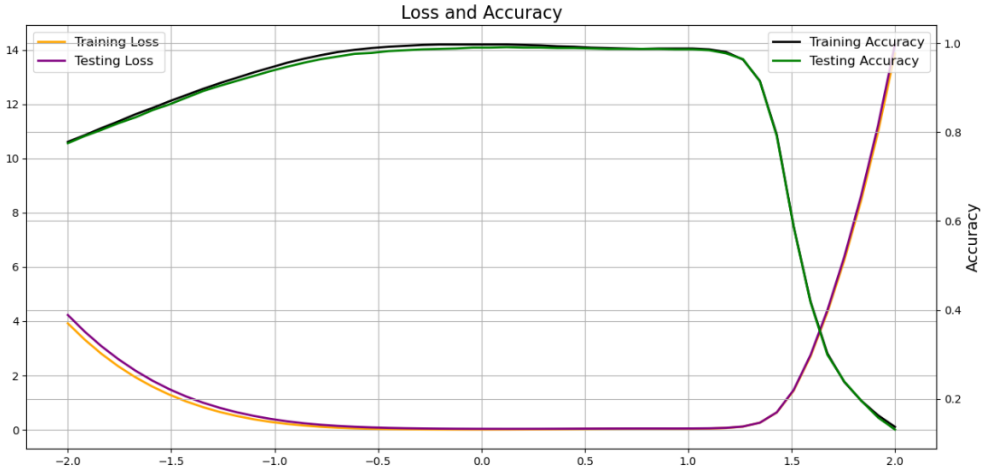
The graph below shows a contrast between the training loss and testing loss in relation to the alpha value.



The graph below compares the testing accuracy and training accuracy as function of the alpha value.



The graph below compares the testing loss and training loss with testing accuracy and training accuracy as function of the alpha value.



Review:

We see that for the models with different learning rates, the accuracy of both training and testing reduces to about 1.5. On the other hand, the graphs with an alpha value of 1.5 show an increase in accuracy.