Homework 1-1: Deep and shallow networks report

For this part of the assignment, I built three neural networks with about the same number of parameters (~2,600) but different depths:

- Shallow network: 2 hidden layers with 50 neurons each
- Deep network: 5 hidden layers with 28 neurons each
- Very deep network: 8 hidden layers with 22 neurons each

I trained each model for 3,000 epochs on the data $x \in [-2,2]$to learn the function $f(x) = \frac{\sin(5\pi x)}{5\pi x}$.

As shown in Figure 1-1, all three models learned the function successfully. The loss curves (left side) show that the very deep model started converging the fastest, and all models reached very low loss values (around $10^{-4}$and 10^{-5}).

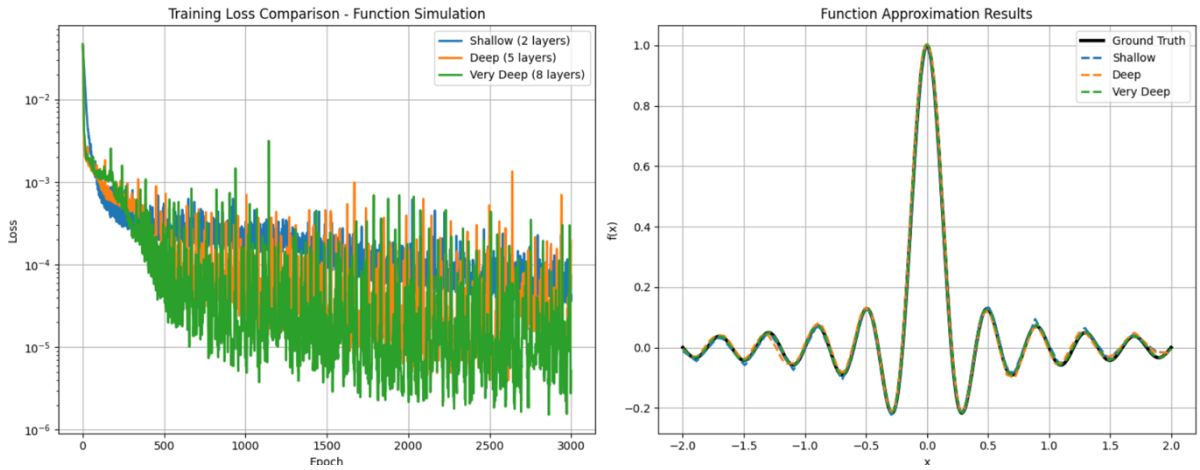


Figure 1-1: Training loss comparison and Function approximation results

The final mean squared error (MSE) values were:

- Shallow network: 0.000054
- Deep network: 0.000153
- Very deep network: 0.000004

The function approximation plot (right side) shows that all three models captured the oscillating shape of the target function, with the very deep model providing the most accurate match across the domain.

Interestingly, the shallow model performed better than the 5-layer deep network, even though we often assume that "deeper is better." However, the very deep 8-layer gave the best results overall. It looks like there might be an optimal depth somewhere in between: too few layers may limit the model's capacity, while too many layers can make training harder — but when trained successfully, very deep networks can still give the most accurate results.

MNIST Classification Results

For the second part of HW1-1, I trained two CNNs on the MNIST dataset:

- Shallow CNN: 2 convolutional layers with 454,922 parameters
- Deep CNN: 6 convolutional layers with 146,938 parameters

Both models were trained for 20 epochs using the Adam optimizer (learning rate = 0.001), batch size = 128, and standard MNIST normalization.

As shown in Figure 1-2, both models converged very quickly, reaching over 98% accuracy by epoch 5. The shallow CNN started with a higher initial accuracy and maintained slightly better performance throughout training.

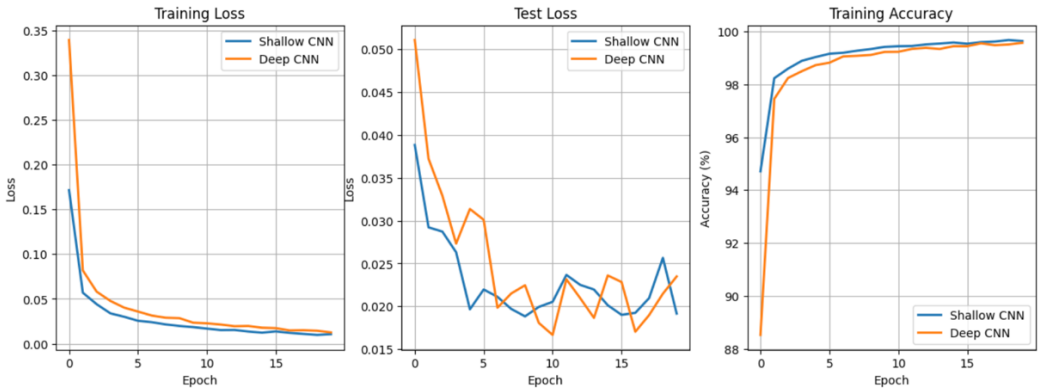


Figure 1-2: Training loss comparison, Test loss and Training accuracy for shallow and deep CNNs

The results:

- Shallow CNN: 99.54% training accuracy, 99.47% test accuracy
- Deep CNN: 99.44% training accuracy, 99.41% test accuracy

```
In [32]: results_mnist = {}
         for model, name in models_mnist:
             results_mnist[name] = train_and_evaluate_mnist(model, name, epochs=20)

Training Shallow CNN...
Parameters: 454922
Epoch 0: Train Acc: 94.71%, Test Acc: 98.74%
Epoch 5: Train Acc: 99.16%, Test Acc: 99.30%
Epoch 10: Train Acc: 99.44%, Test Acc: 99.35%
Epoch 15: Train Acc: 99.54%, Test Acc: 99.47%

Training Deep CNN...
Parameters: 146938
Epoch 0: Train Acc: 88.52%, Test Acc: 98.22%
Epoch 5: Train Acc: 98.82%, Test Acc: 98.98%
Epoch 10: Train Acc: 99.23%, Test Acc: 99.49%
Epoch 15: Train Acc: 99.44%, Test Acc: 99.41%
```

The training and test loss curves show stable learning without overfitting. The accuracy curves confirm that both models generalize well.

Interestingly, the shallow CNN, despite having fewer layers but more total parameters, slightly outperformed the deeper CNN. This suggests that for a relatively simple dataset like MNIST, wider layers can be more effective than simply stacking more layers.

Homework 1-2: Optimization

In this part of the homework, I explored how the network's weights, gradient norms, and loss values evolve during training. I used the small CNN defined earlier, trained it on MNIST for 8 epochs, and repeated the experiment 8 times with different random initializations. During training, I saved the model weights every 3 epochs, recorded gradient norms at each iteration, and tracked the loss values.

Figure 2-1 shows the PCA projection of all saved weight snapshots over the course of training. Each point represents the network's parameters at a given snapshot, and its color indicates accuracy. We can see that each of the 8 runs followed a slightly different path through weight space, but all trajectories eventually moved toward regions associated with high accuracy (yellow points near the end). This confirms that even though different random initializations lead to different local minima, many of these minima achieve similarly good performance on MNIST.

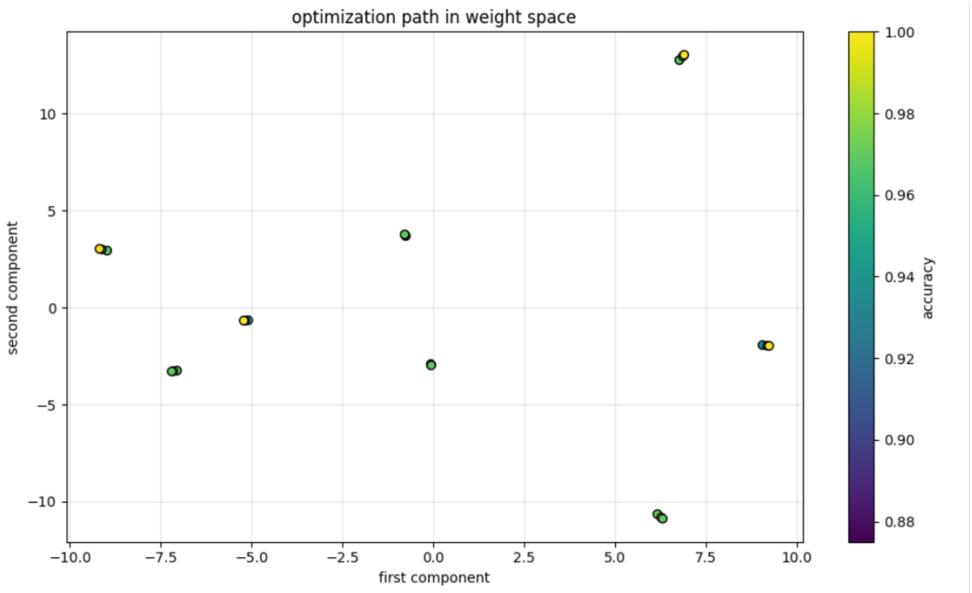
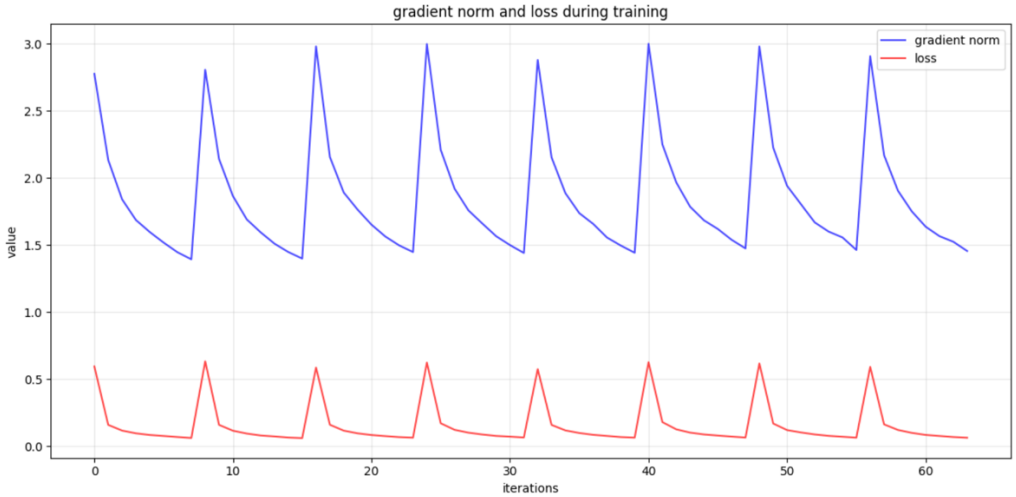


Figure 2-1: Weight space optimization path (color = accuracy)

Figure 2-2 plots the gradient norm (blue) and loss (red) across training iterations. The loss decreases steadily over time, while the gradient norm starts large and gradually becomes smaller as the model approaches convergence. The "saw-tooth" pattern in the gradient norm is due to epoch restarts; gradients briefly rise at the beginning of each epoch and then decay. This behavior is expected and shows that the network is making consistent progress toward a minimum without instability.




```
length mismatch: weights=24, losses=22
```

Figure 2-2: Gradient norm and loss per iteration

Figure 2-3 shows the relationship between the L2 norm of the weights and the loss at each saved snapshot. Most points cluster in a narrow band of weight norms between about 17.9 and 18.3, and within this band, the loss stays consistently low. This suggests that the model finds a relatively flat region of the loss landscape, meaning there are many parameter settings with similar performance, which is a good sign for generalization.

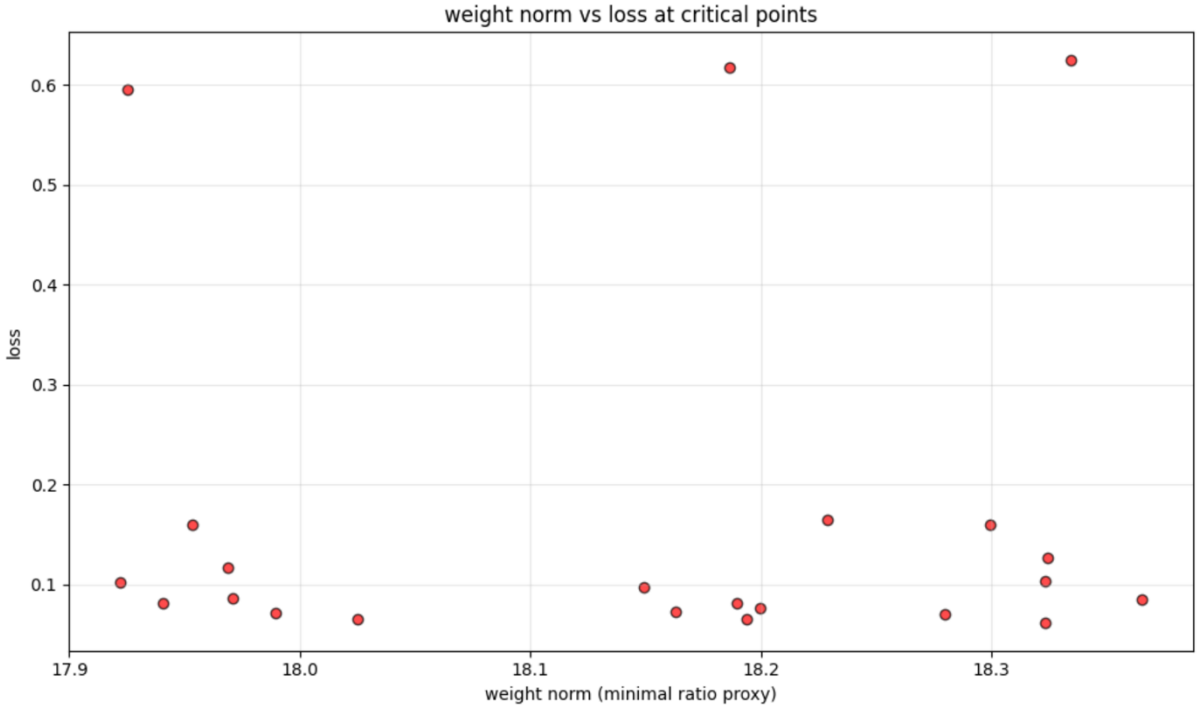


Figure 2-3: Weight norm and loss at snapshots

Homework 1-3: Generalization

I randomized all training and test labels and trained five fully connected models with parameter counts ranging from 1,675 → 41,435. Training was run for 50 epochs using Adam with a learning rate of 0.001 and a batch size matched to each model's setup.

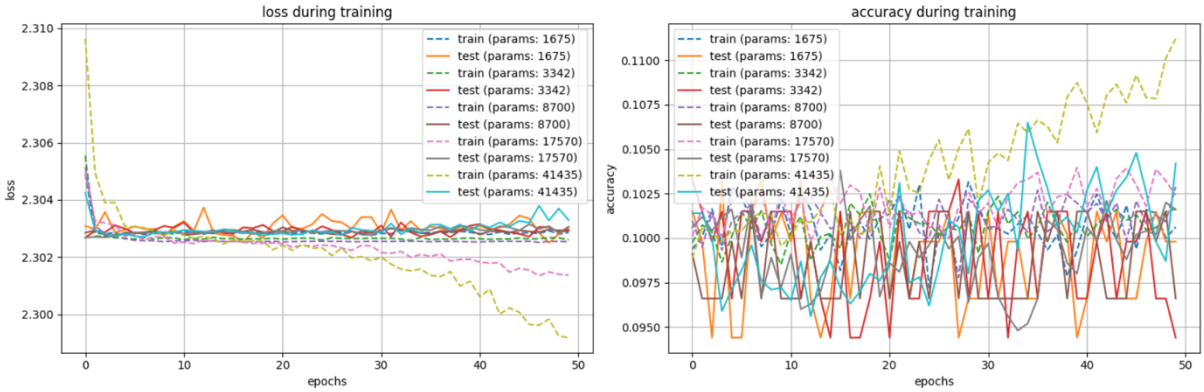


Figure 3-1: Training and test loss curves and the accuracy curves

All models quickly converged to a loss near 2.303, which is the expected cross-entropy for random guessing over 10 classes. In addition, accuracy stayed around 10% (chance level) for both train and test sets. The largest model (41k parameters) showed a slight upward trend in training accuracy near the end, suggesting it started to memorize some random labels, but test accuracy stayed near 10%.

Based on my 50-epoch runs, the networks did not fully memorize random labels**.** If we trained much longer or used an even larger network, we would expect training accuracy to eventually approach 100%, but test accuracy would stay at chance because the labels carry no real pattern to generalize from.

Finally, I measured sensitivity (average gradient norm) across five batch sizes: 5, 25, 125, 500, 1500**.**

As shown in Figure 3-2, sensitivity is highest with very small batches (batch size 5), drops to its minimum near batch size 125, and then slowly increases again for very large batches.

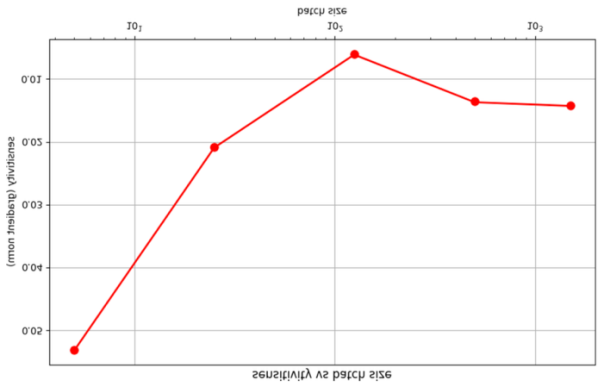


Figure 3-2: Sensitivity and batch size

This result is consistent with the idea that:

- Small batches introduce more noise (high gradient norms).
- Large batches can lead to sharper minima.
- Medium batch sizes find flatter minima (lower sensitivity), which is often associated with better generalization.