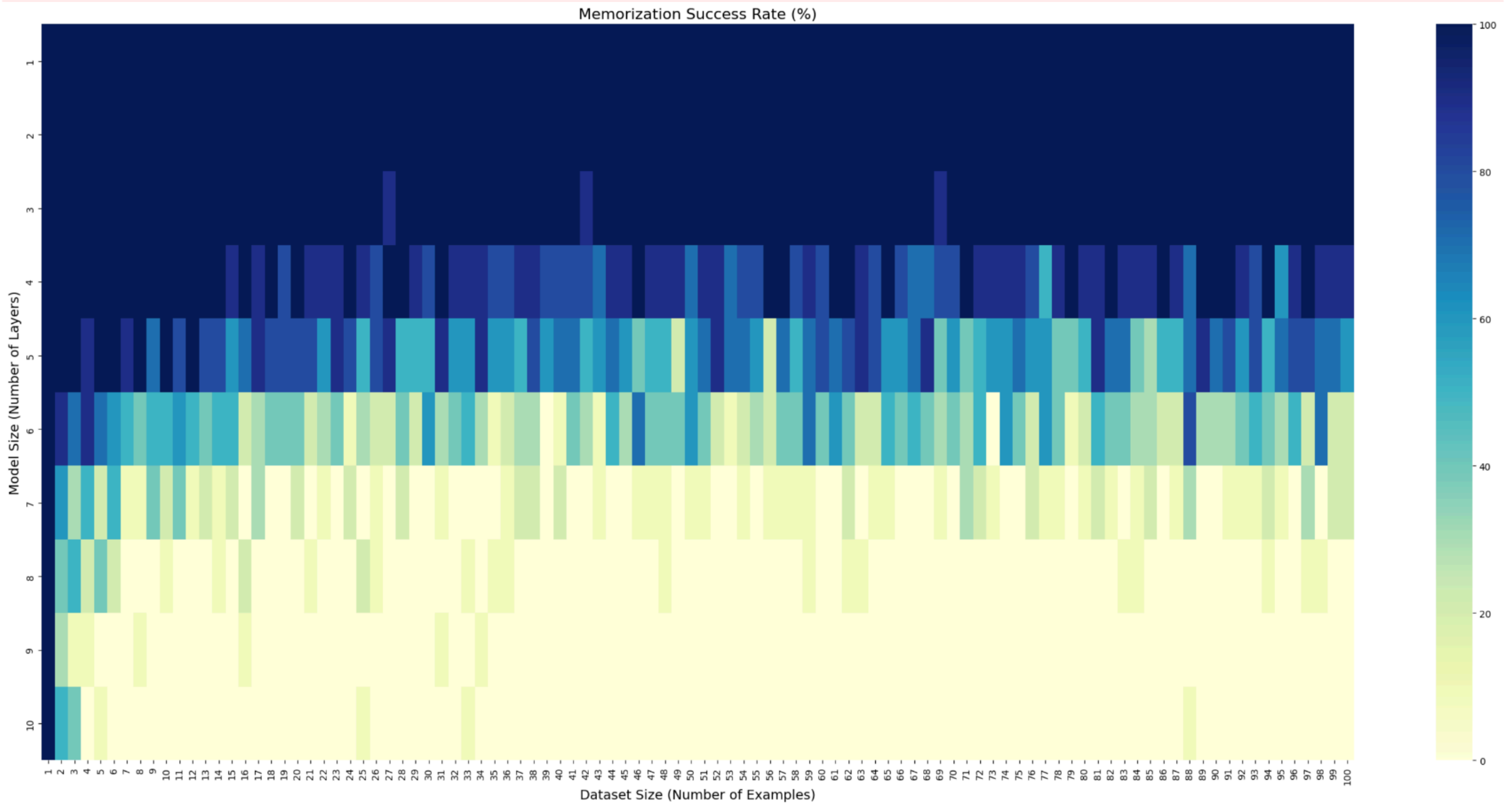
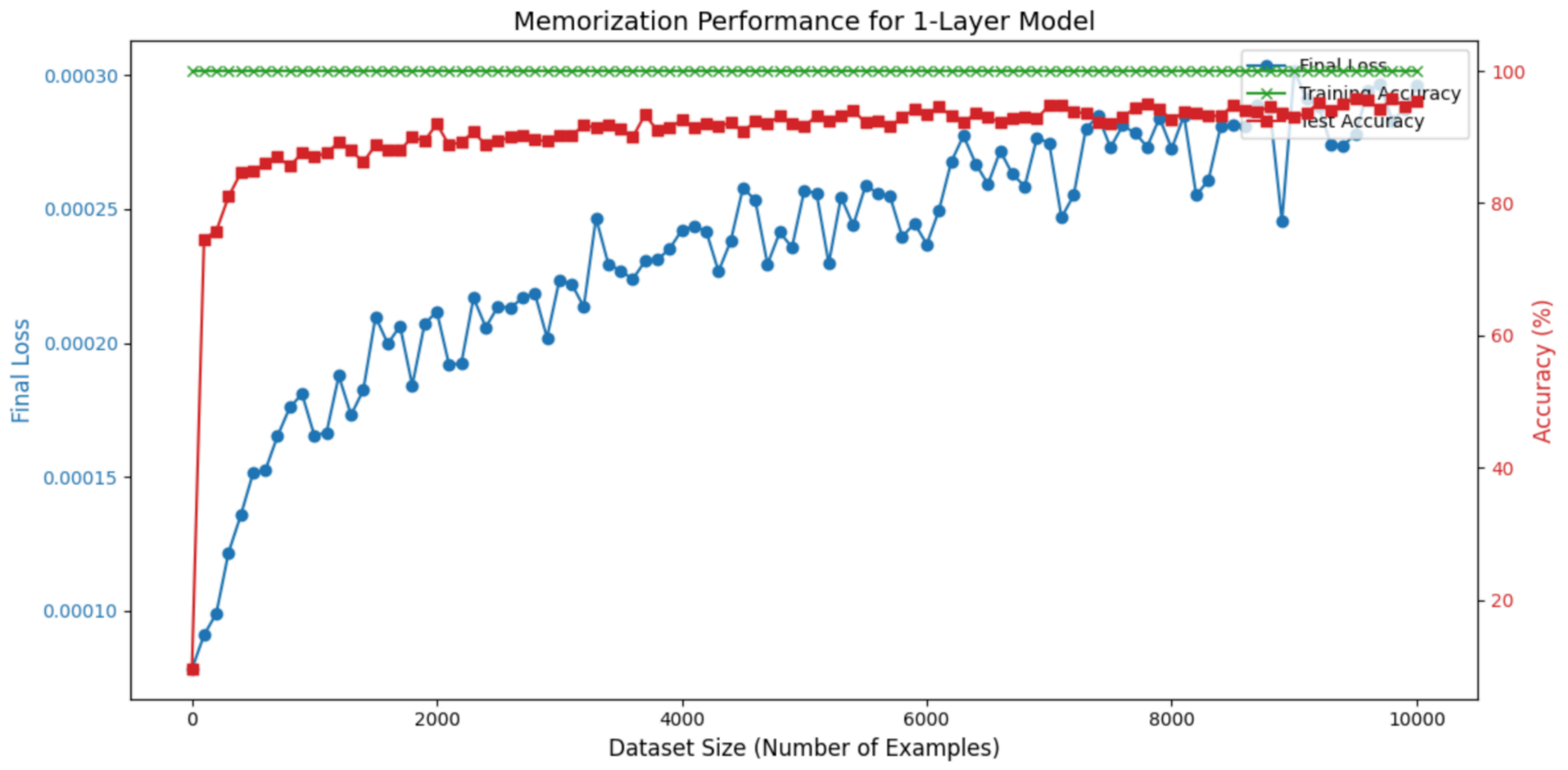
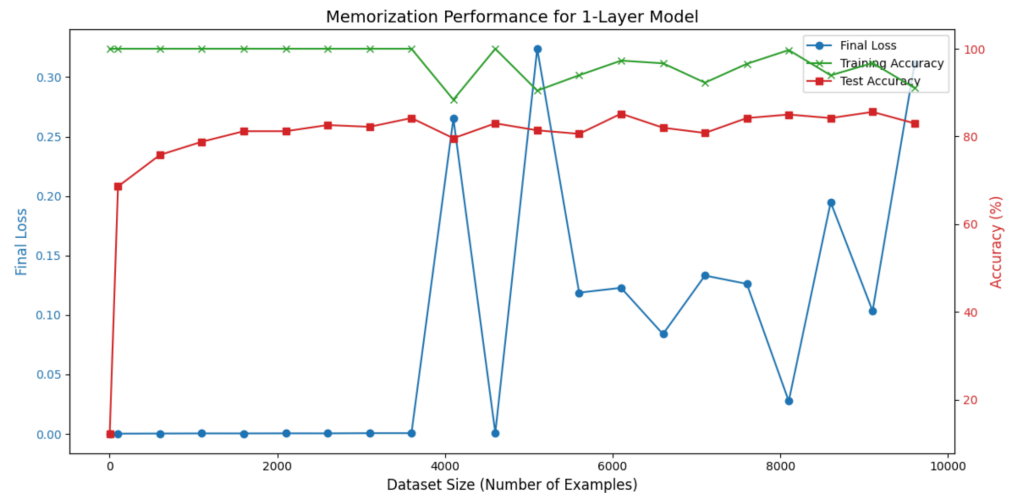
10-18-24: can we learn a new back propagation method?
Can you generate tons of models that memorizes a small part of information and then make another model that takes in the entire dataset and generates the model's parameters?

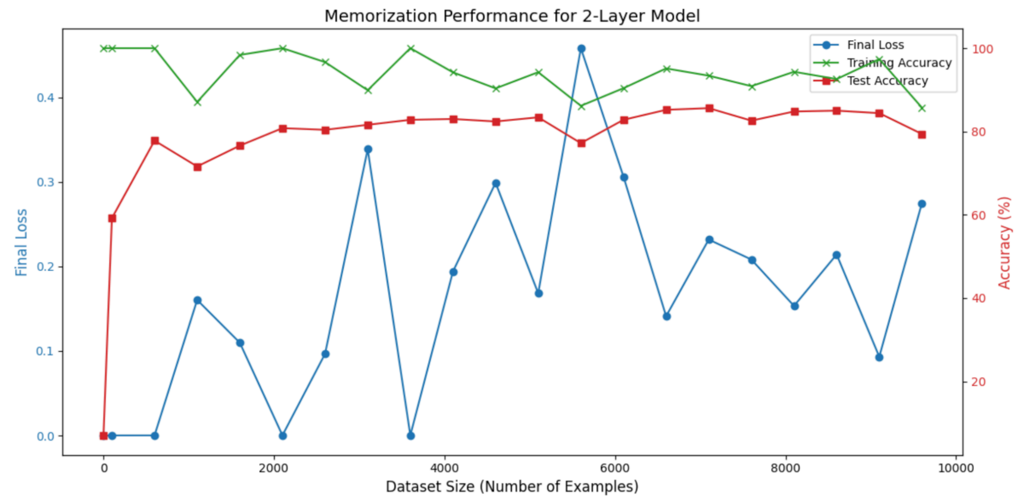


Testing the 1-2 layer model for memorization and test score accuracy on held out data. Mnist is pretty simple so test accuracy is really high. (2 layer is basically the same, hidden behind this)



Fashionmnist is a slightly harder dataset that gives more interesting results.





Predicting model:
- I can get high training accuracy but test level generalization is very low.
- This may be due to every model I train also being perfectly overfitted, perhaps generalized models would be more accurate

```
          +---------------------------------------------------+
          |                  Input Data                       |
          |    (Batch Size, Subset Size * Per Example Size)   |
          |                Shape: (B, 100 * 794)              |
          +---------------------------------------------------+
                         |
                         v
              +----------------------------+
              |      Split into Subsets     |
              |    Each Subset Shape: (B, 794) |
              +----------------------------+
                         |
                         v
+---------------------------+   +---------------------------+
|    Embedding Branch #1     |  |    Embedding Branch #2     |
|     Input: (B, 794)        |  |     Input: (B, 794)        |
|     Output: (B, 256)       |  |     Output: (B, 256)       |
+---------------------------+   +---------------------------+
                         .
                         .
              (One Embedding Branch per Example)
                         .
                         .
+---------------------------+   +---------------------------+
|    Embedding Branch #100   |  |      Embedding Branch #n    |
|     Input: (B, 794)        |  |     Input: (B, 794)        |
|     Output: (B, 256)       |  |     Output: (B, 256)       |
+---------------------------+   +---------------------------+
                         |
                         v
            +-----------------------------------------+
            |        Stack Embeddings Together        |
            |   Output Shape: (100, B, 256)           |
            +-----------------------------------------+
                         |
                         v
            +-----------------------------------------+
            |    Attention-Based Fusion Layer         |
            |  Inputs: (100, B, 256)                  |
            |  Output: (100, B, 256)                  |
            +-----------------------------------------+
                         |
                         v
            +-----------------------------------------+
            |      Mean Pool Over Example Dimension   |
            |   Output Shape: (B, 256)                |
            +-----------------------------------------+
                         |
                         v
            +-----------------------------------------+
            |      Parameter Generation Network       |
            |   Inputs: (B, 256)                      |
            |   Output: (B, 6370)                     |
            | (Predicted Parameters for Target Model) |
            +-----------------------------------------+
                         |
                         v
           +------------------------------------------+
           |  Predicted Parameters for SimpleNN Model  |
           |  Output Shape: (B, 6370)                  |
           +------------------------------------------+
```

10-19-24: adversarial training random
Randomizing parameters internally per step improves adversarial resilience?

- Base line training of model with nothing else:
    ~70% accurate after 30 epochs
    Adversarial Accuracy on Baseline Model: 16.14%

- Random neuron with final adversarial test: 0.01% randomizer neurons per step:
    71% accuracy
    19% accuracy adversarial

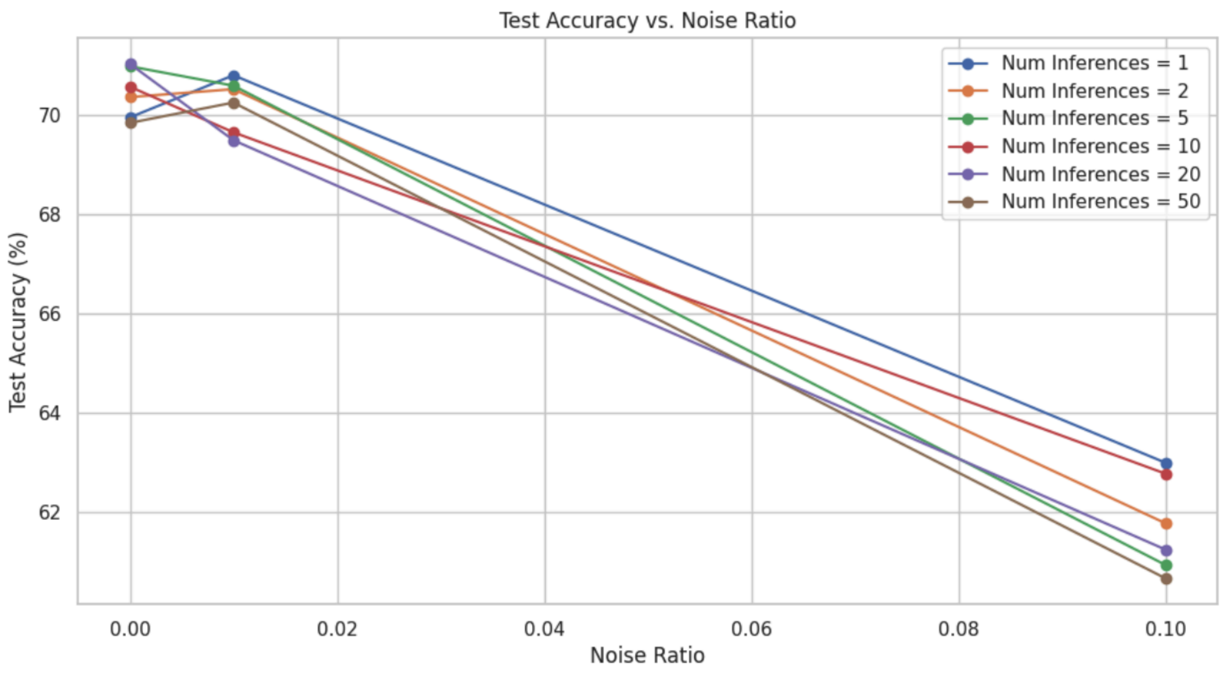
- Adversarial training standard:
    Clean Accuracy: 40.18%
    Adversarial Accuracy: 32.29%

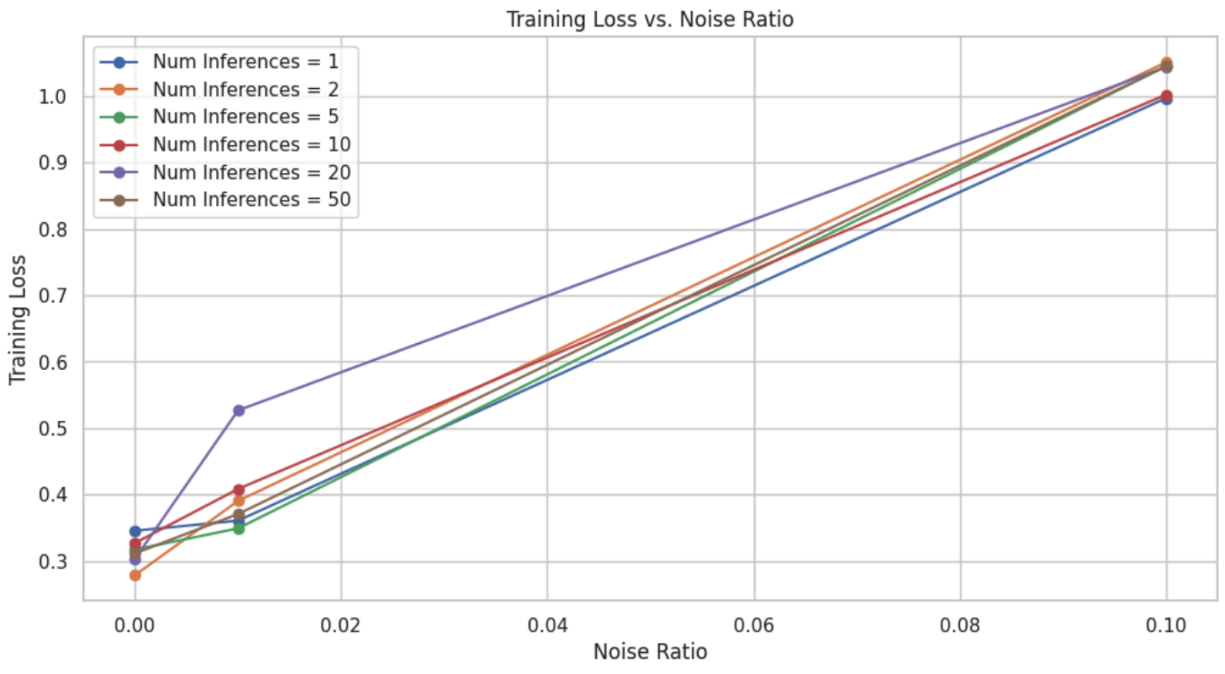
- Random neuron trained with adversarial:
    Best Clean Accuracy: 27.32%
    Best Adversarial Accuracy: 23.98%

10-21-24: Random noise injections and averaging multiple inferences:

Train 10 epochs standard the special inference:

```
Noise Ratio: 0.0, Num Inferences: 1, Test Loss: 0.9594, Accuracy: 72.97%
Noise Ratio: 0.0, Num Inferences: 10, Test Loss: 0.9594, Accuracy: 72.97%
Noise Ratio: 0.0, Num Inferences: 100, Test Loss: 0.9594, Accuracy: 72.97%
Noise Ratio: 0.0, Num Inferences: 1000, Test Loss: 0.9594, Accuracy: 72.97%
Noise Ratio: 0.001, Num Inferences: 1, Test Loss: 0.9586, Accuracy: 72.98%
Noise Ratio: 0.001, Num Inferences: 10, Test Loss: 0.9588, Accuracy: 72.96%
Noise Ratio: 0.001, Num Inferences: 100, Test Loss: 0.9588, Accuracy: 72.96%
Noise Ratio: 0.001, Num Inferences: 1000, Test Loss: 0.9588, Accuracy: 72.97%
Noise Ratio: 0.1, Num Inferences: 1, Test Loss: 1.2435, Accuracy: 61.61%
Noise Ratio: 0.1, Num Inferences: 10, Test Loss: 1.1853, Accuracy: 62.91%
Noise Ratio: 0.1, Num Inferences: 100, Test Loss: 1.1810, Accuracy: 63.03%
Noise Ratio: 0.1, Num Inferences: 1000, Test Loss: 1.1808, Accuracy: 62.87%
Noise Ratio: 0.2, Num Inferences: 10, Test Loss: 1.6942, Accuracy: 44.98%
Noise Ratio: 0.2, Num Inferences: 100, Test Loss: 1.6756, Accuracy: 45.37%
Noise Ratio: 0.2, Num Inferences: 1000, Test Loss: 1.6749, Accuracy: 45.50%
Noise Ratio: 0.2, Num Inferences: 5000, Test Loss: 1.6745, Accuracy: 45.52%
Noise Ratio: 0.5, Num Inferences: 10, Test Loss: 3.0416, Accuracy: 16.40%
Noise Ratio: 0.5, Num Inferences: 100, Test Loss: 2.9588, Accuracy: 17.09%
Noise Ratio: 0.5, Num Inferences: 1000, Test Loss: 2.9499, Accuracy: 17.33%
Noise Ratio: 0.5, Num Inferences: 5000, Test Loss: 2.9498, Accuracy: 17.36%
```





10-22-24: NN based Gradient updates.
Gradients in back prop are averaged. What if instead of doing this we implemented something more complex that would be a NN that would take in the batch size of gradients and with that select or produce a better update value. Then after all of the neurons chose this update the model would do a second back prop step to give an error signal to these gradient update neurons.

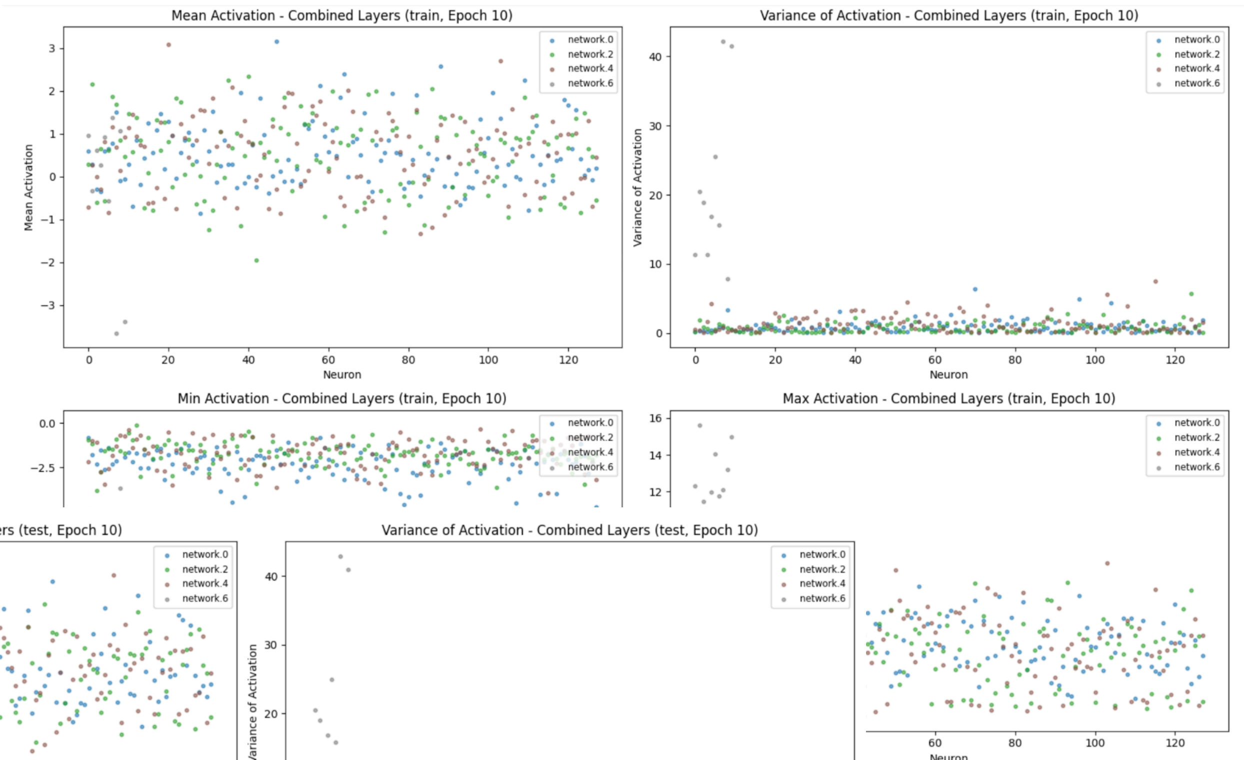
10-22-24: How do neurons hidden state behave over a training sequence

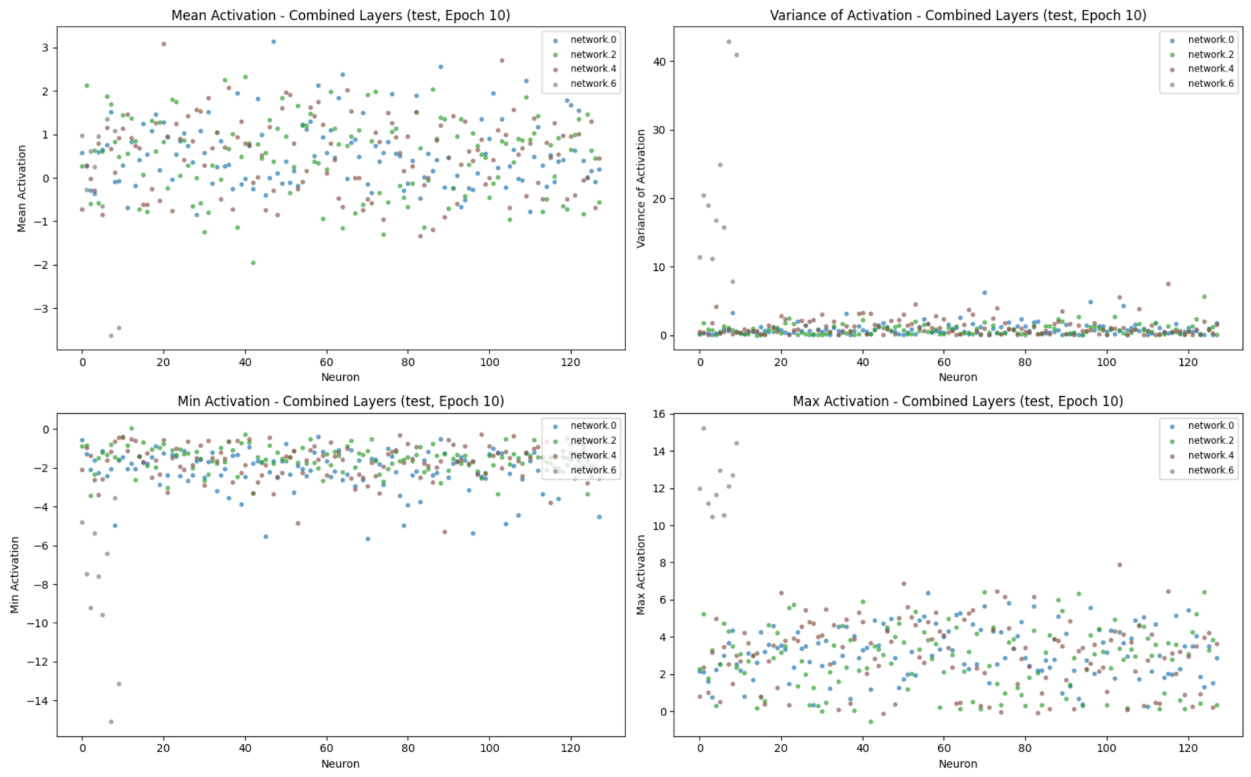
- train 1 epoch
- inference the entire dataset
- Capture the hidden states of every single neuron over the entire dataset for this 1 epoch of trained model
- You then can calculate your analysis statistics for these.

Basic MNist seemed to overfit almost instantly for some layers. I switched to fashionmnist and results are far closer to the expected.

There are some neurons that increase variance but overall it is pretty stable over the epochs.

There is a slight difference between the training and testing in the variance values but it is very minor implying good fit. Final layer still has very high variance, train vs test is basically the same.





Increase depth and number of epochs, looking for overfitting and depth effect.

Didn’t see any of the effect the standard mnist did for training instability over time. Training and test seemed basically the same.

Variance clearly decreases to a stable amount then increases over time, in the individual plots, not sure why the cumulative one does not show this