Neural Learning Answer Set 3

German Shiklov
317634517

Problem 1.2:

In the first simulation, we've shown a straightforward train / test losses with accuracy, for the default values of the network:

```
layers_sizes = [784, 32, 10] # flexible, but must be [784,...,10]
epochs = 4       # number of times to repeat over the whole training set
eta = 0.1        # learning rate
batch_size = 30 # number of samples in each training batch
```

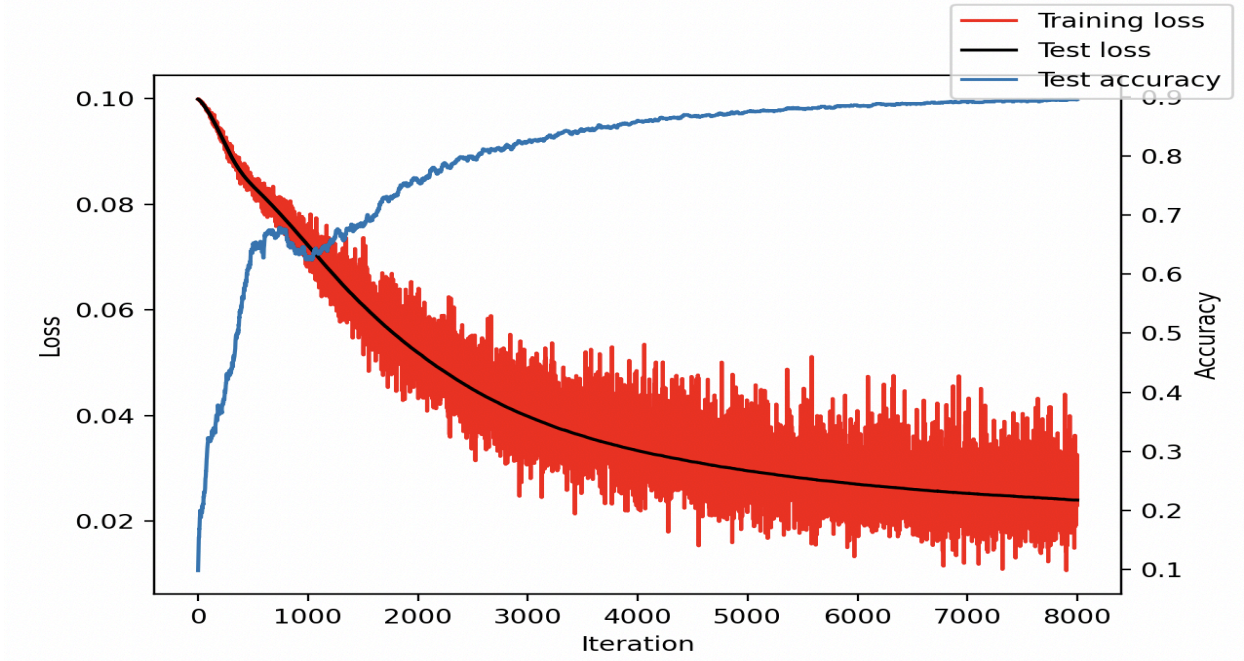


Figure 1. Simulation 1.

The misclassified digits have a very low perplexing effect for the human eye.

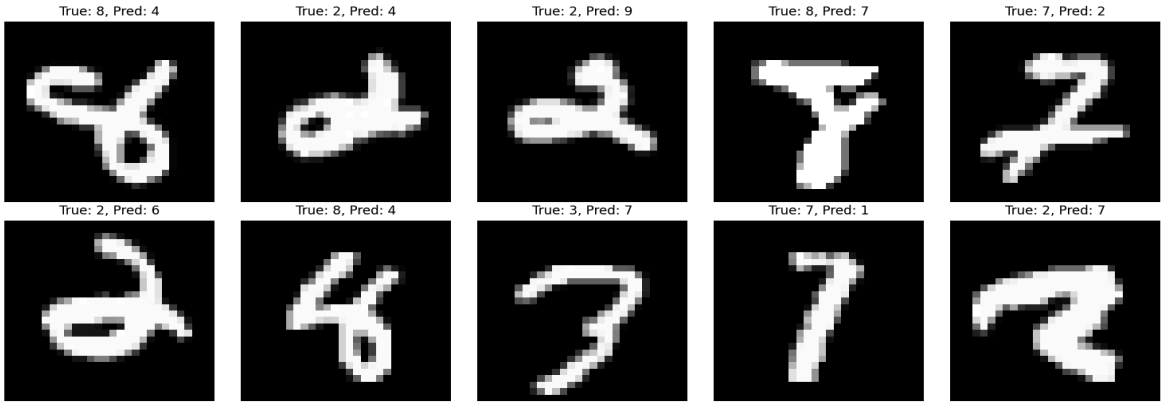


Figure 2. Simulation of wrong classification for digits.

The following simulation 2 is from parameters,

```
layers_sizes = [784, 64, 10]
epochs = 4
eta = 0.1
batch_size = 32
```

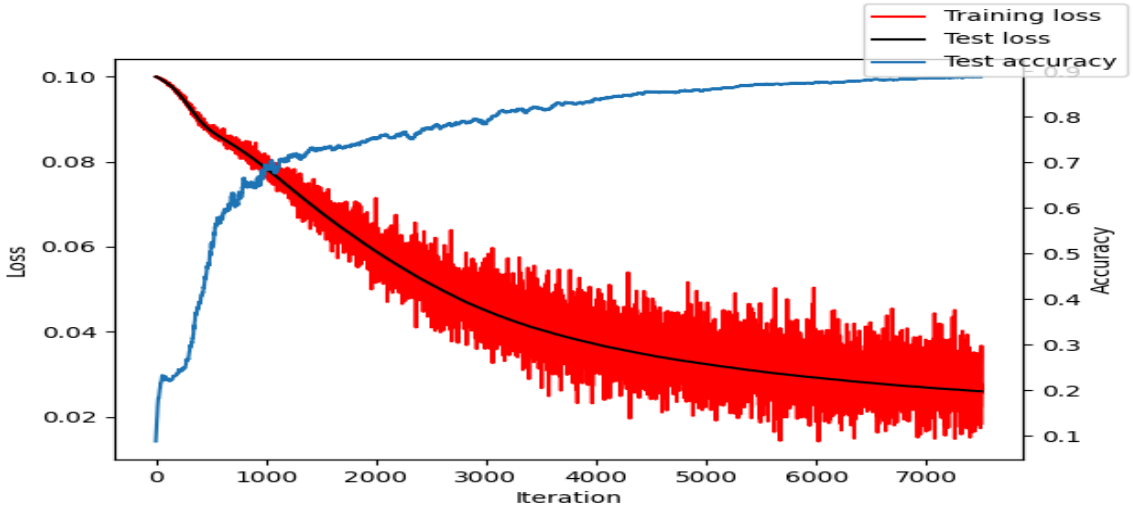


Figure 3. Simulation 2; BS:32; LR:0.1; EPOCHES: 4; HL: 64.

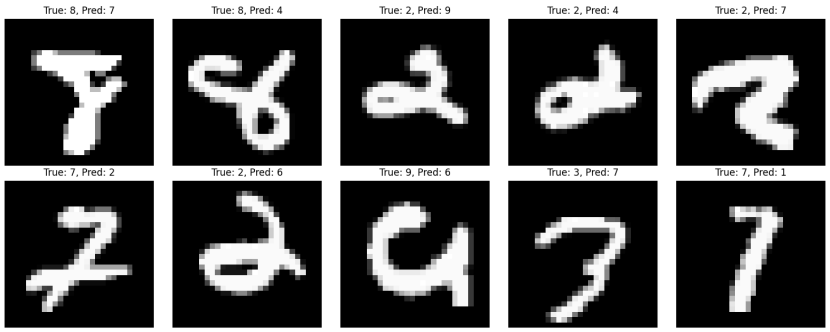


Figure 4. Simulation 2; Incorrect 10 digits classification.

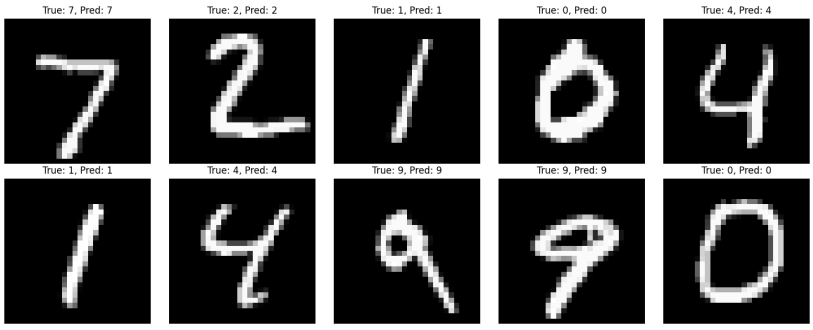


Figure 5. Simulation 2; Correct classification of digits.

(1.2.1)

Figure 3 shows that the accuracy has almost reached 0.9 for this amount of iterations. In figure 4 we observe the misclassified digits which to a human eye is “easier” to tell as we have much more accuracy

in our self human model then this network trained network. Only the last option, right bottom corner, may be confusing the observer as 1 rather than 7. Less ambiguous to humans.

Throughout figures 6 to 9, we observe learning rates of 0.01 and 1.

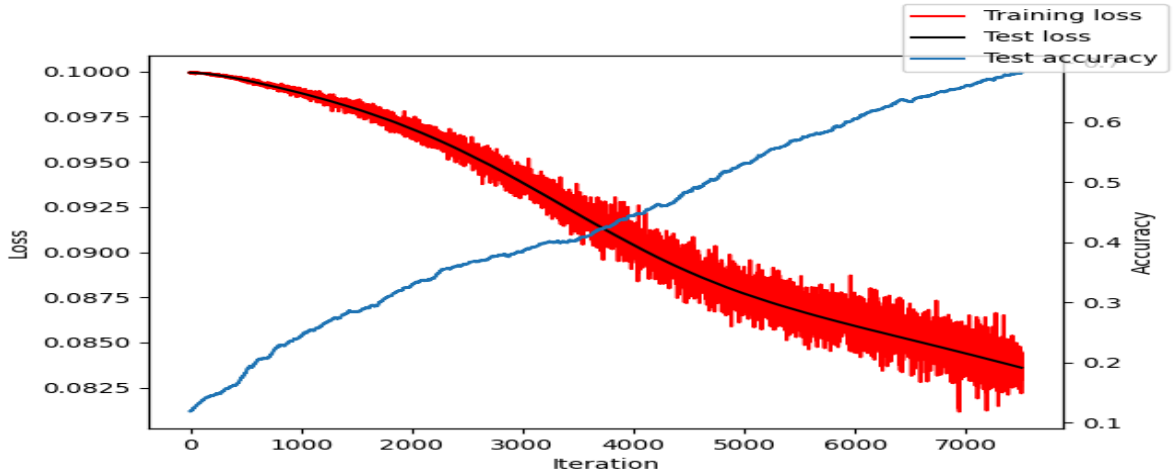


Figure 6. Simulation 3. BS:32; LR:0.01; EPOCHES: 4; HL: 64.

Incorrect classification

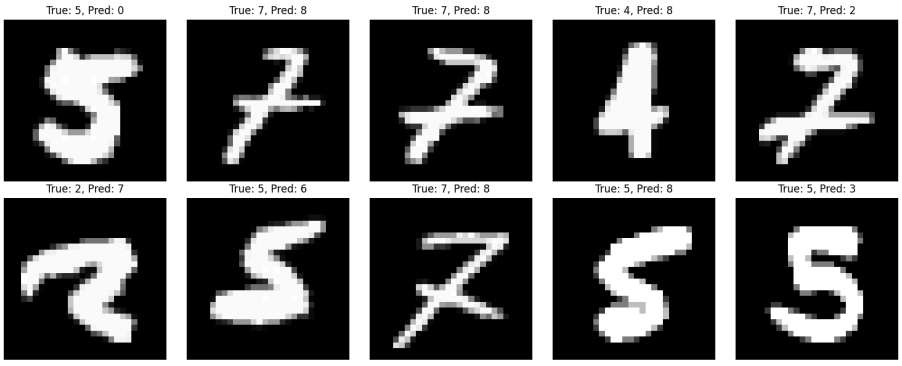


Figure 7. Simulation 3, misclassified digits.

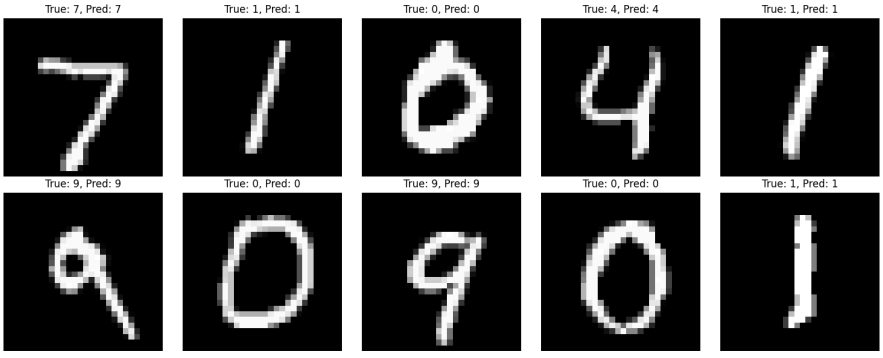


Figure 8. Simulation 3. Correct classification.

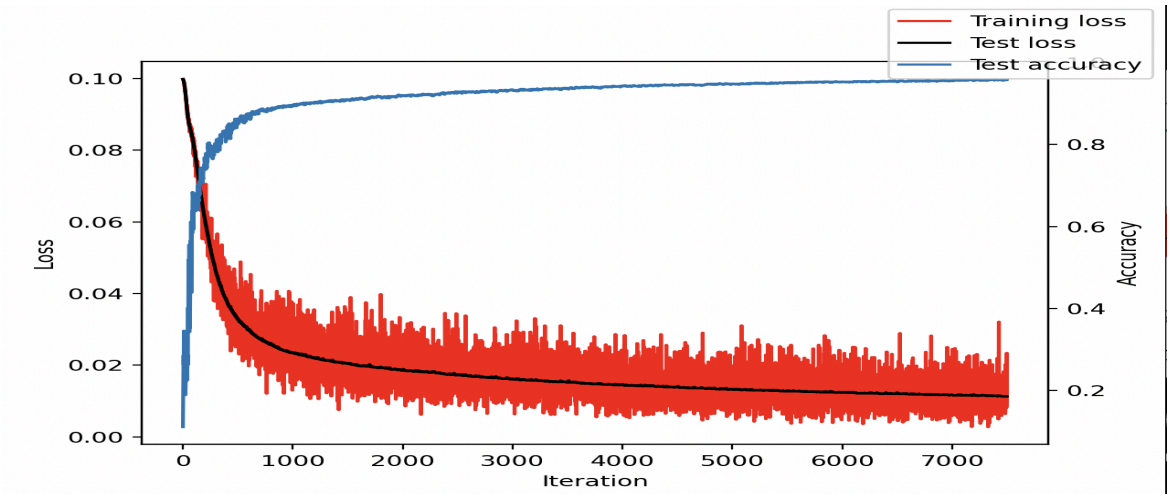


Figure 9. Simulation 3. LR: 1; BS: 64; EPOCHS: 4.

(1.2.2)

With a learning rate of 0.1, the optimization process may converge quickly, but it's also prone to overshooting and instability, especially if not carefully tuned. In our case we have quite enough examples, yet of course the accuracy is not 1. (see Figure 3).

A learning rate of 0.01 is moderate, implying smaller updates to the model parameters compared to a learning rate of 0.1. This can lead to more stable training dynamics and smoother convergence.

While a lower learning rate generally results in slower convergence, it can also help the optimization process to avoid overshooting or getting stuck in local minima.

A learning rate of 1 is considered very high. It implies very large updates to the model parameters during each iteration of optimization. That said, the optimization process is likely to be highly unstable. The updates may oscillate wildly, and the loss function may fail to converge to a good solution, yet Figure 7 shows the model has converged to what seems like 95% accuracy, and almost zero loss. In most cases, a learning rate of 1 is too large for effective training, and it often leads to divergence or poor performance of the model.

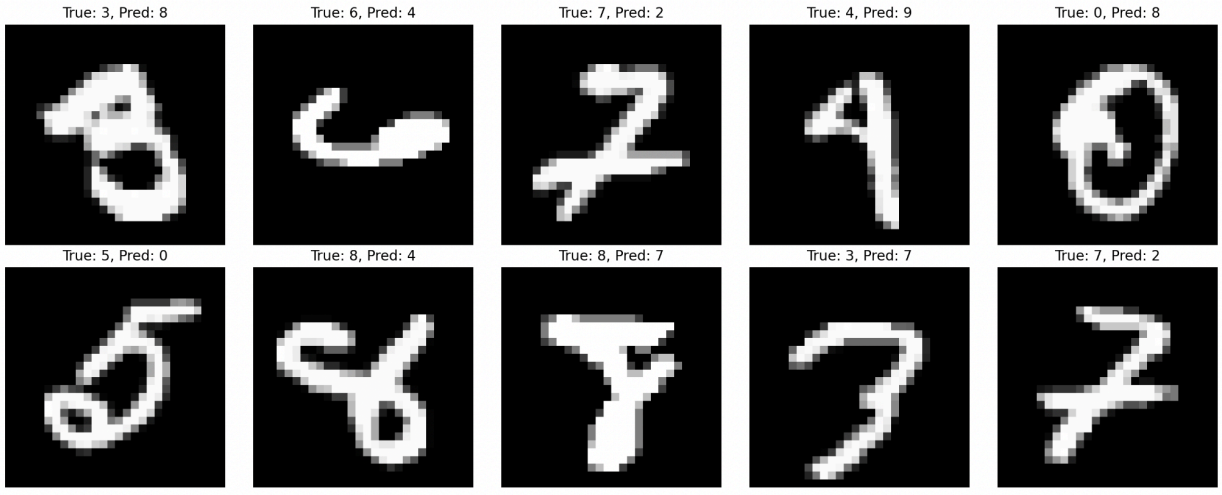


Figure 10. Simulation 3. Misclassified digits. True 4, Pred 9 - indeed looks like 4.

(1.2.3.a)

Training a model on such a small and uniformly distributed subset of the MNIST dataset does not truly match the real-world statistics of handwritten digits. In a more naturally distributed dataset, some digits may be more frequent than others, and the variability within each class of digit would be greater. A small uniform sample might miss out on this variability and could bias the model toward the specific examples it was trained on, reducing its generalizability. The model might overfit to the training data and not perform well on a more statistically representative test set. It's important to have a training set that captures the diversity and distribution of the data the model is expected to work with after deployment. Basically this approach helps in avoiding biases towards more frequently occurring digits as everything is uniform.

(1.2.3.b)

Based on the results of the uniform distribution for digits, the more we would have let it run, it would have learned better the dataset - might of course have an overfit due to such fact. Yet, in this simulation it looks alright to give it more runtime.

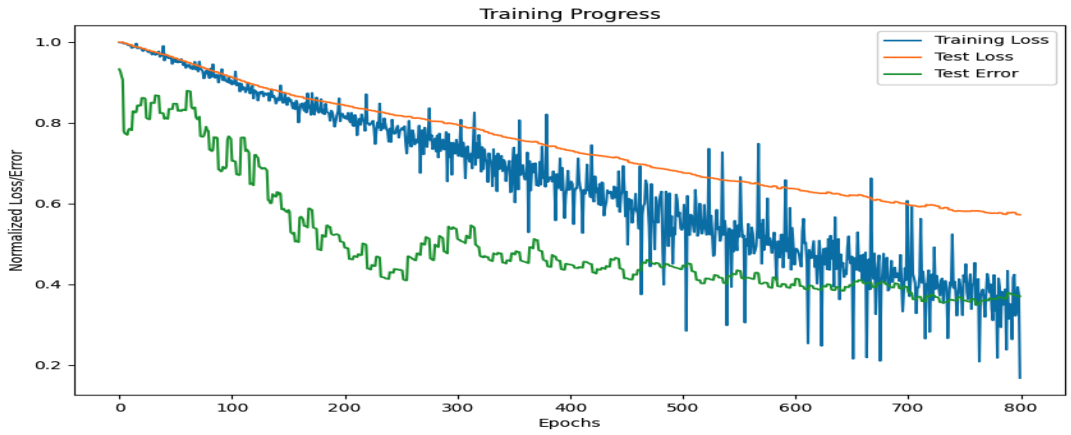


Figure 11. Simulation 4. Uniform distribution of digits, subset of 10 examples.

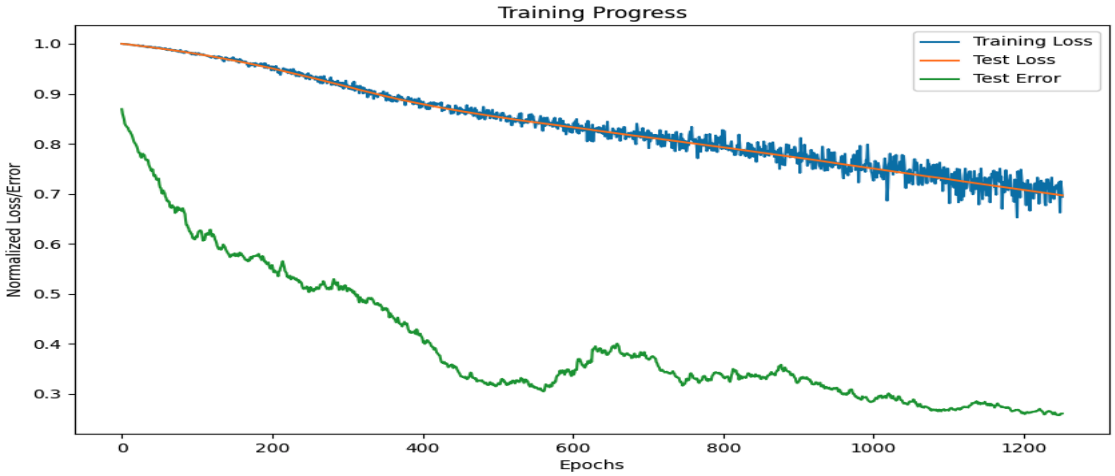


Figure 12. Simulation 5. Uniform distribution of digits, subset of 1000 examples.

(1.2.4)

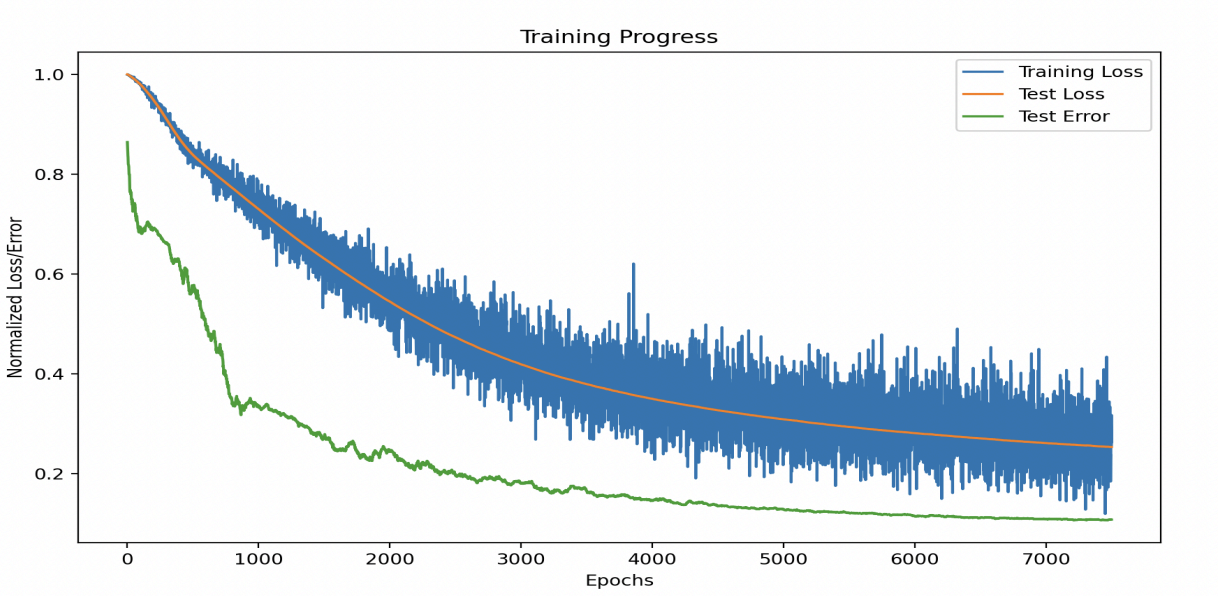
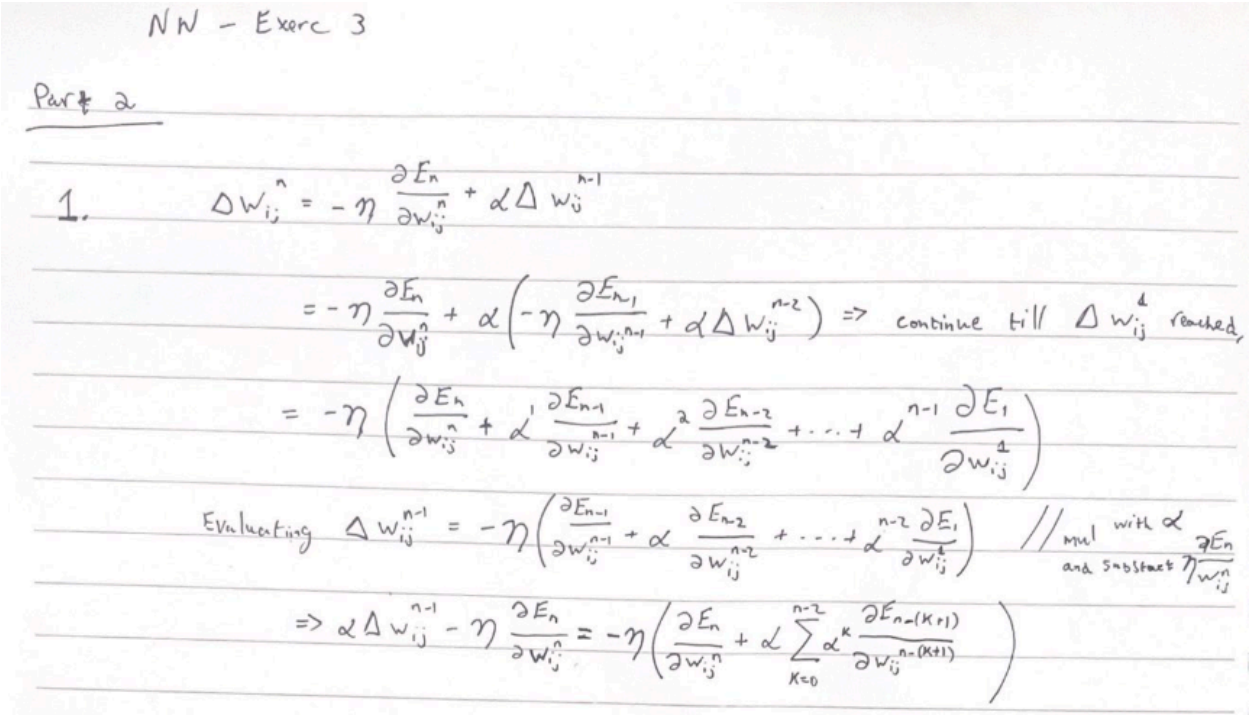


Figure 13. Simulation 6. Epochs: 4. BS: 32; HL: 32. LR: 0.1. All Samples.

This time, in Figure 9 the Test Error hits a plateau of roughly 0.1, where it cannot minimize the error below. More likely that in this experiment, an early stop at 4k iterations would be sufficient to prevent the overfitting. Unlike in Figure 8 and 9, where the plateau is not visible and thus leading to potentially more iterations to improve the model's accuracy.

Problem 2:



2. It's seen that for $\{\alpha | 0 \leq \alpha < 1\}$, is a neceessary condition as the gradients diminish over time, whereas $\alpha > 1$ would most likely grow infinitely and won't provide convergence.

3. Generally, if the gradient $\frac{\partial E_n}{\partial w_{ij}}$ keeps the same sign over iterations

Will increase the updates because of past gradients accumulation. Such behavior may lead to faster convergence, while consistently moving towards a minimum.

* When the sign of the gradient flips, the momentum helps to "cancel" the update, as the previous momentum term won't let it to make a huge step, but rather small updates (steps)

* This momentum's behavior helps to prevent updates which are too reactive for a gradient calculation, therefore we'll get a smoother trajectory towards a minimum, and less oscillation.