AssignmentReport-Group1

February 16, 2022

1 Assignment 2 Report

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet. If you are not familiar with markdown.

Before delivery, **remember to convert this file to PDF**. You can do it in two ways: 1. Print the webpage (ctrl+P or cmd+P) 2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc`.

2 Task 1

2.1 task 1a)

$$w_{ji} = w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}} = w_{ji} - \alpha \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}} = w_{ji} - \alpha \delta_j \cdot x_i$$

$$\delta_k^i = \frac{\partial C}{\partial z_k^i} = \sum_k \frac{\partial C}{\partial z_k^{i+1}} \cdot \frac{\partial z_k^{i+1}}{\partial z_j^i} = \sum_k \delta_k^{i+1} \cdot f'(z_j^i) \cdot w_{kj}^i = f'(z_j^i) \cdot \sum_k \delta_k^{i+1} \cdot w_{kj}^i$$

2.2 task 1b)

Defining:

I: number of input nodes J: number of nodes in hidden layer K: number of output nodes

$W_1 = [w_{ji}]$ (shape: $J \times I$)

$W_2 = [w_{kj}]$ (shape: $K \times J$)

$\curvearrowleft$: input (shape: $I \times 1$)

$F_{\curvearrowleft} = W_1 \cdot \curvearrowleft$ (shape: $J \times 1$)

$\eth_{\curvearrowleft} = f(F_{\curvearrowleft})$ (shape: $J \times 1$)

$F_{\not\curvearrowleft} = W_2 \cdot \eth_{\curvearrowleft}$ (shape: $K \times 1$)

$\hat{\curvearrowright} = f(F_{\not\curvearrowleft})$: output (shape: $K \times 1$)

$\delta_{\curvearrowleft} = [\delta_j]$ (shape: $J \times 1$)

$\delta_{\not\curvearrowleft} = [\delta_k]$ (shape: $K \times 1$)

Update rules:

$W_1 := W_1 - \alpha \delta_{\not\perp}{}^T \cdot \curvearrowleft$

$W_2 := W_2 - \alpha \delta_{\not\perp}{}^T \cdot \supset\!\!\!\!\!\!\perp$

3 Task 2

3.1 Task 2a)

3.1.1 Training:

Mean: 33.55274553571429

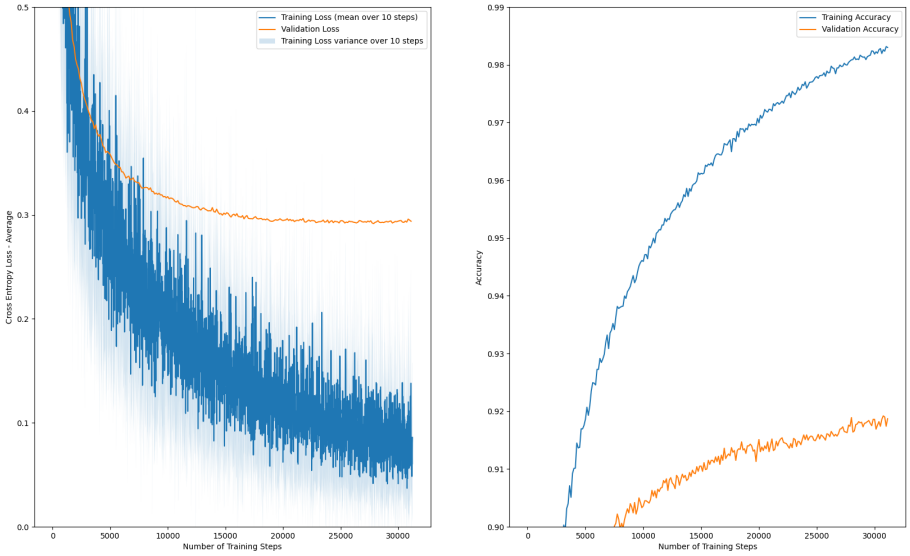
Standard deviation: 78.87550070784701

3.1.2 Validation:

Mean: 33.791224489795916

Standard deviation: 79.17246322228644

3.2 Task 2c)

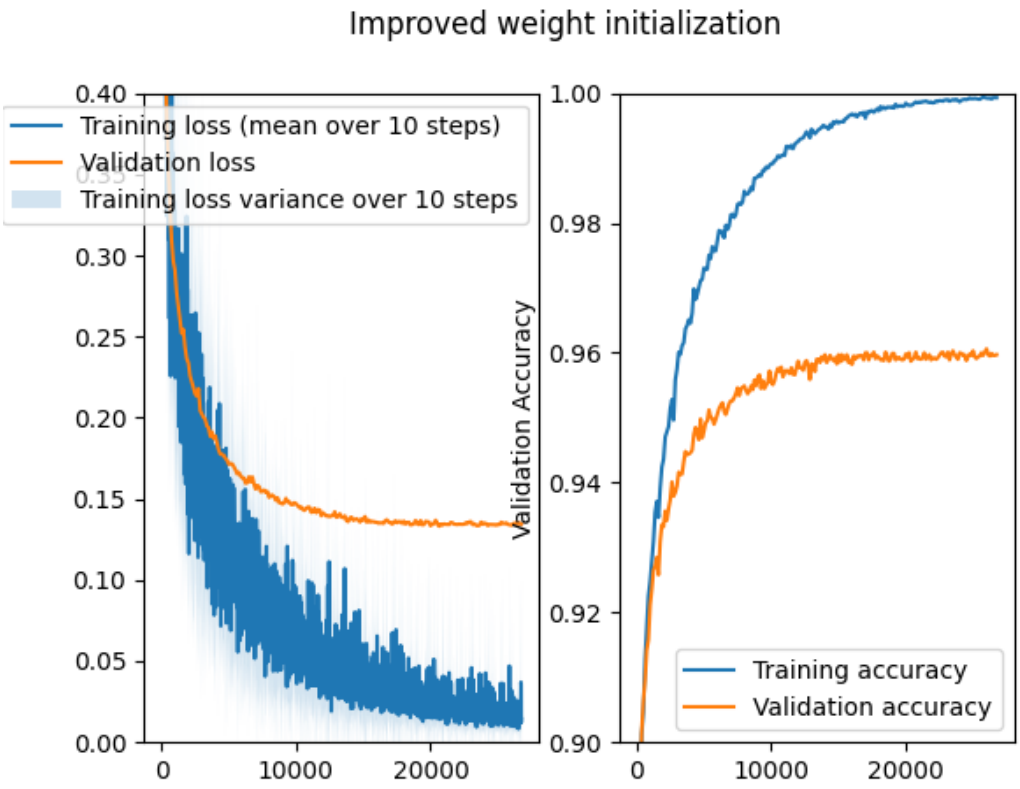


3.3 Task 2d)

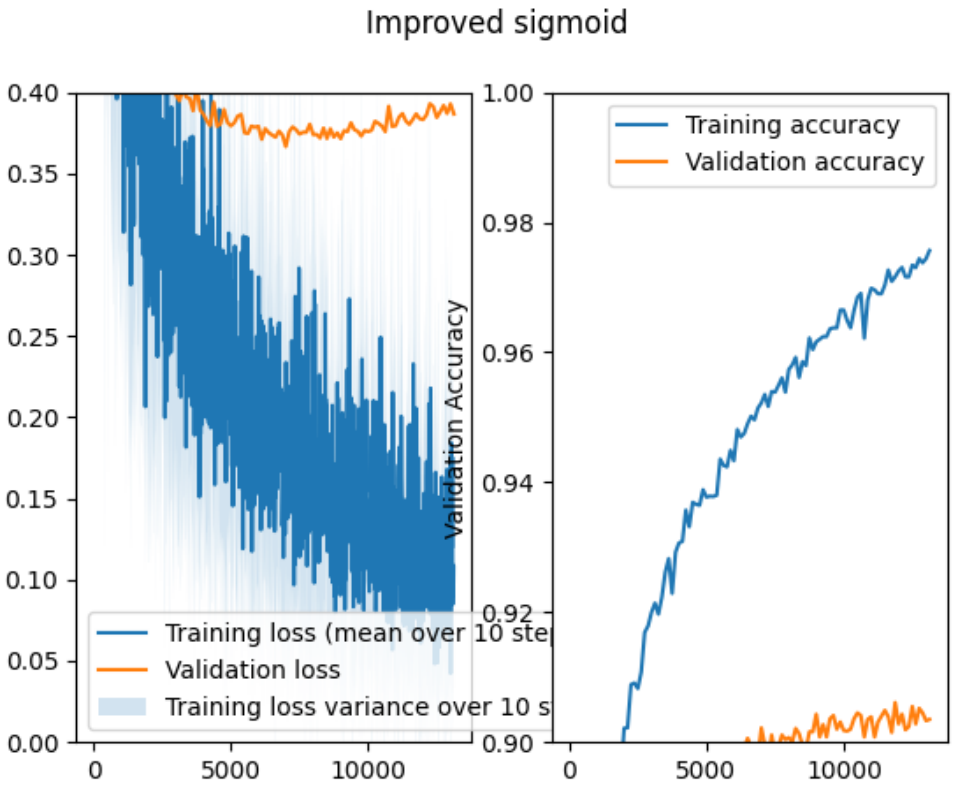
The number of parameteres, consisting of weights and biases, in the network are: $(28 \cdot 28 + 1) \cdot 64 + (64 + 1) \cdot 10 = 50890$

4 Task 3

4.0.1 Improved weigth initialization

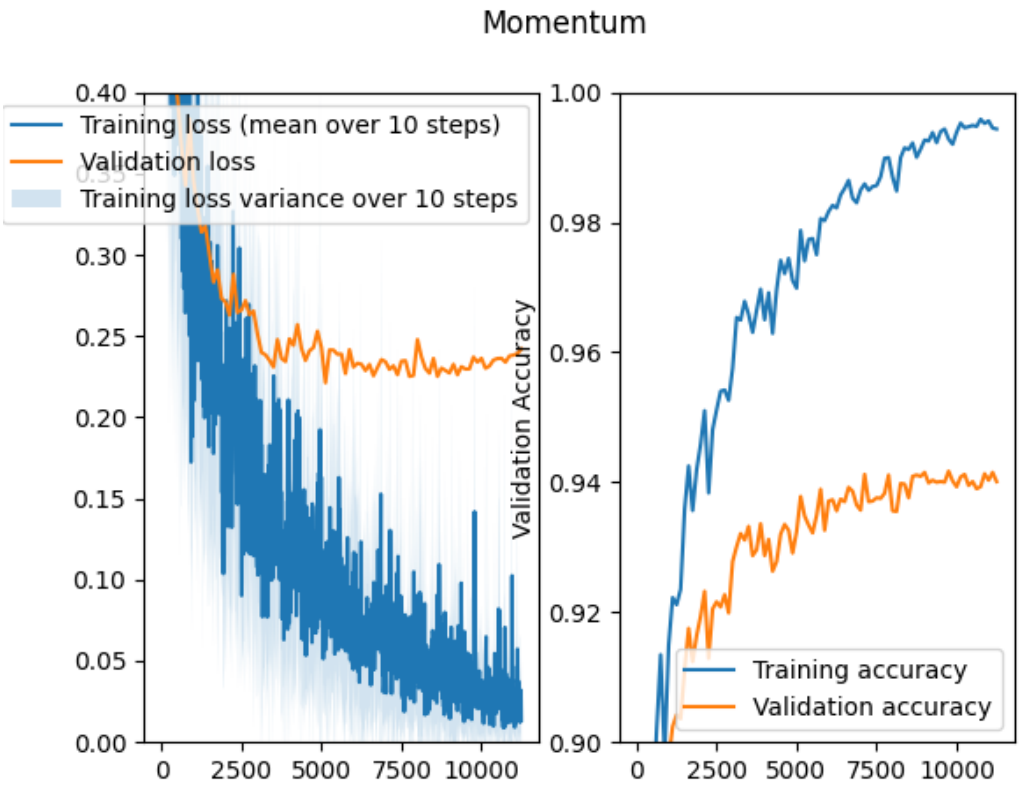


With improved weight initialization the accuracy and loss in both training and validation gets over 90% very fast. In addition the accuracy becomes better at the end as well. However it still took a long time to train using 42 epochs before the early stopping ended the training. ### *Improved*



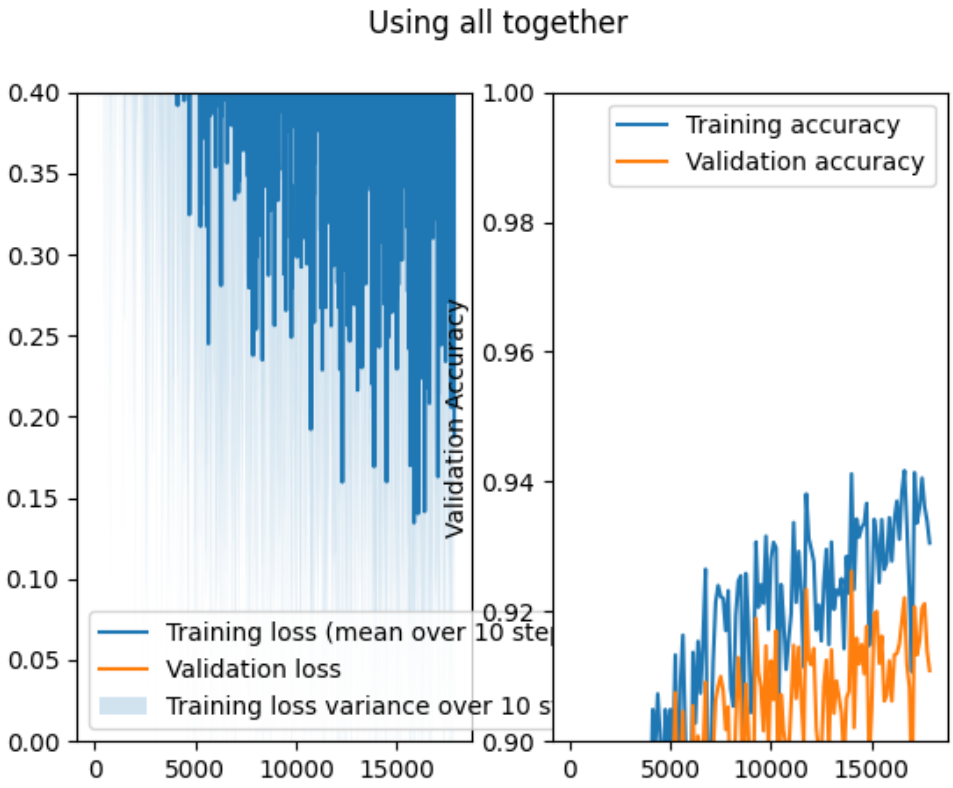
sigmoid

The improved sigmoid function resulted in very high loss and low accuracy. We assume this is because of some error in our code, most likely in the implementation of the improved sigmoid. However, we have asked for help in the lab hours, but they could not find the error either. ### Momentum



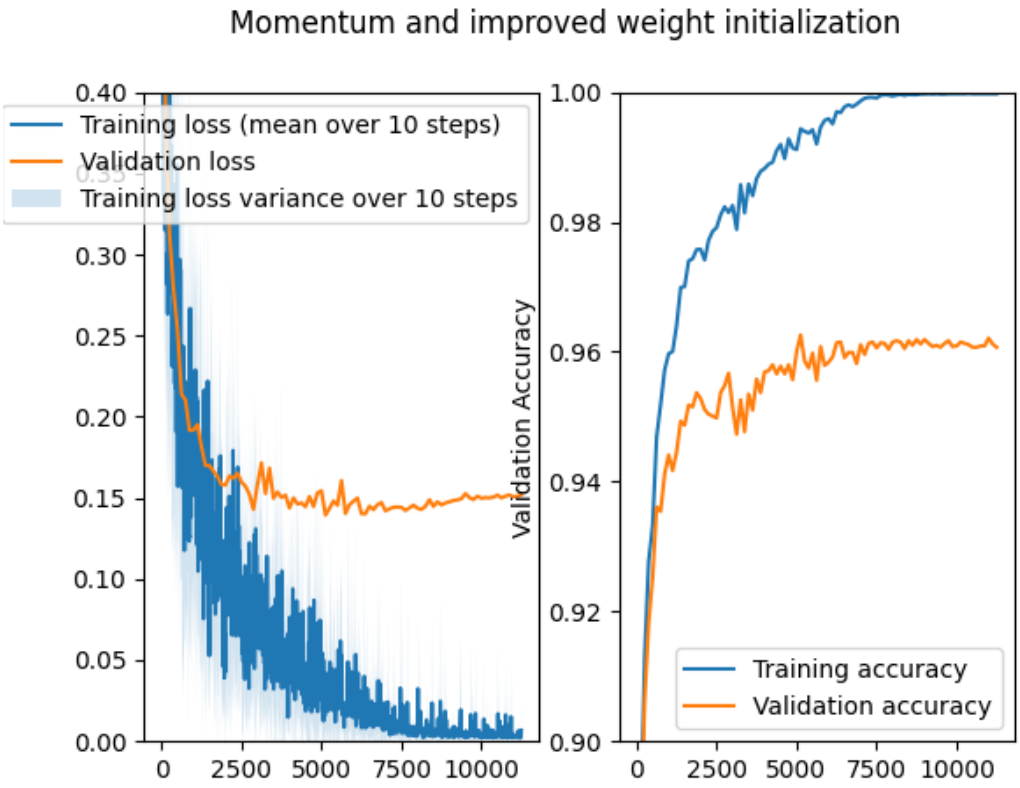
When using momentum, the accuracy and loss of both the training and validation improves fast, but it seems that it is not as fast as the training with improved weight initialization in the beginning. However, it trains really fast towards the end as well stopping after only 18 epochs with very good accuracy (but not as good as with weight init). The loss and accuracy seems also to be more noisy than before. This makes sense since with momentum one is not going directly in the direction of best improvement for this specific step, but (hopefully) in the more general direction of decent of the whole error/loss surface.

4.0.2 Improved weigth initialization, improved sigmoid and momentum



This is really bad probably because the improved sigmoid implementation does not work. Therefore we ended up just using momentum and weight initialization as shown below. This is also what we use in task 4.

4.0.3 Improved weigth initialization and momentum



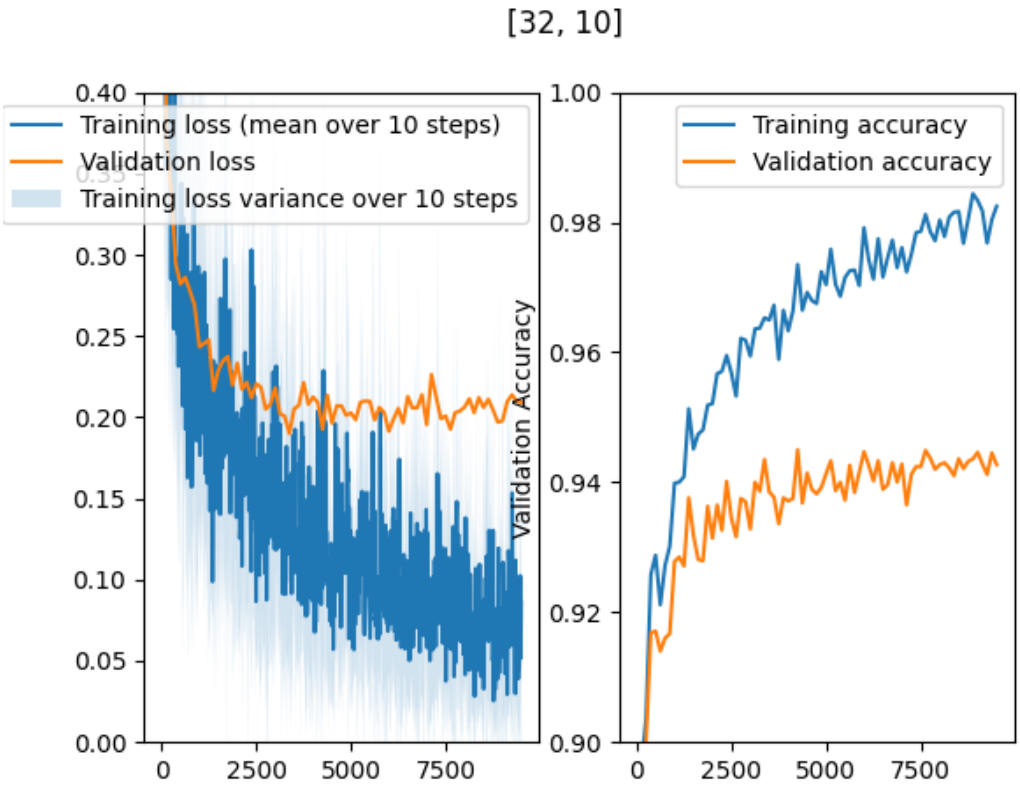
With both improved weight initialization and momentum one can see that it results is something with "the best of both worlds". It improves really fast in the beginning. It seems that a combination of both is better in the beginning compared to the two improvements separately. It also improves fast towards the end as well stopping at epoch 18 similarly to when using only momentum. Also similar to when using momentum, the loss and accuracy improves with more "noise", but it seems that the effect is smaller now than with using only momentum.

The training accuracy ends at 100% this suggests that one might be overfitting to the training set. However, the accuracy on the validation set is also very high so it does not seem that it is having a bad effect.

5 Task 4

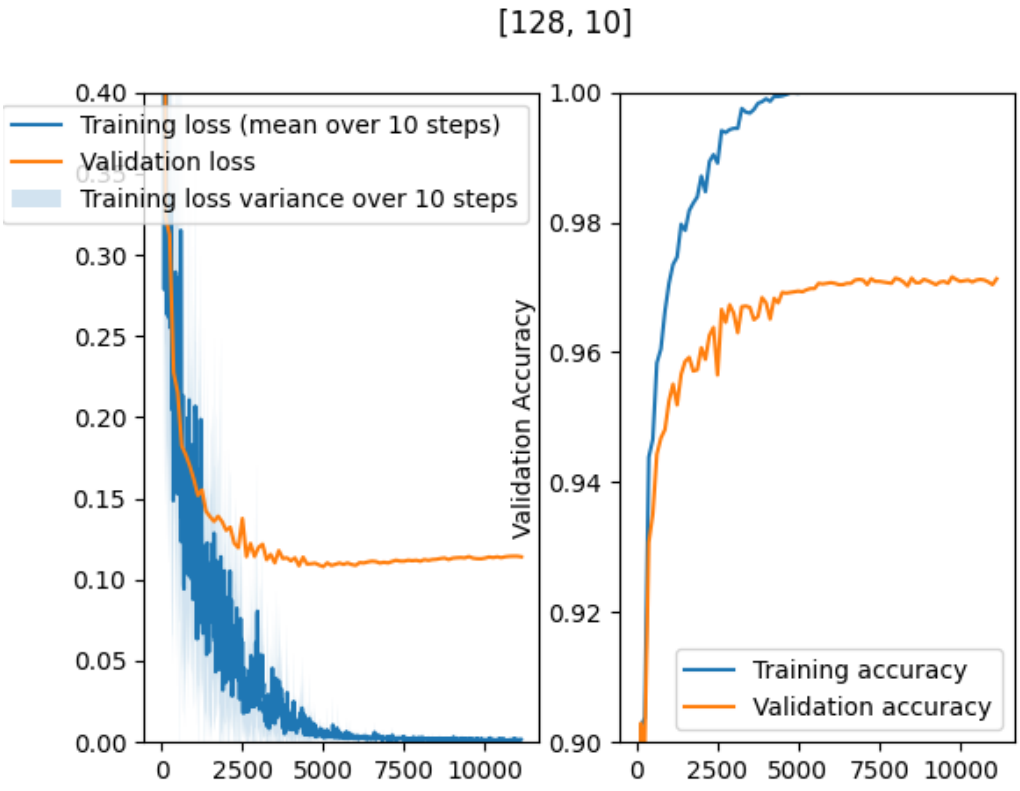
5.1 Task 4a)

With 32 nodes in the hidden layer we see that the loss and accuracy, while not being bad, have become worse than with the original 64 nodes. It converges very quickly and stop training which is good. However, this is at the cost of the accuracy decreasing with ≈ 0.02. The reason for the early stopping could be a result from the algorithm simply reaching its full potential, in regards to performance, which is not as good as previously given the fewer amount of nodes.



5.2 Task 4b)

With 128 nodes in the hidden layer, we see the loss and accuracy improving compared to 64. It reaches the maximum accuracy and minimal loss at an earlier stage than previously, which is a result from having more nodes (i.e. less restrictions). However, the training accuracy is *suspicioulsy* high, similar to the case with 64 nodes. This is an indication of overfitting, which should be handled by either implementing regularization, reducing number of nodes, etc.

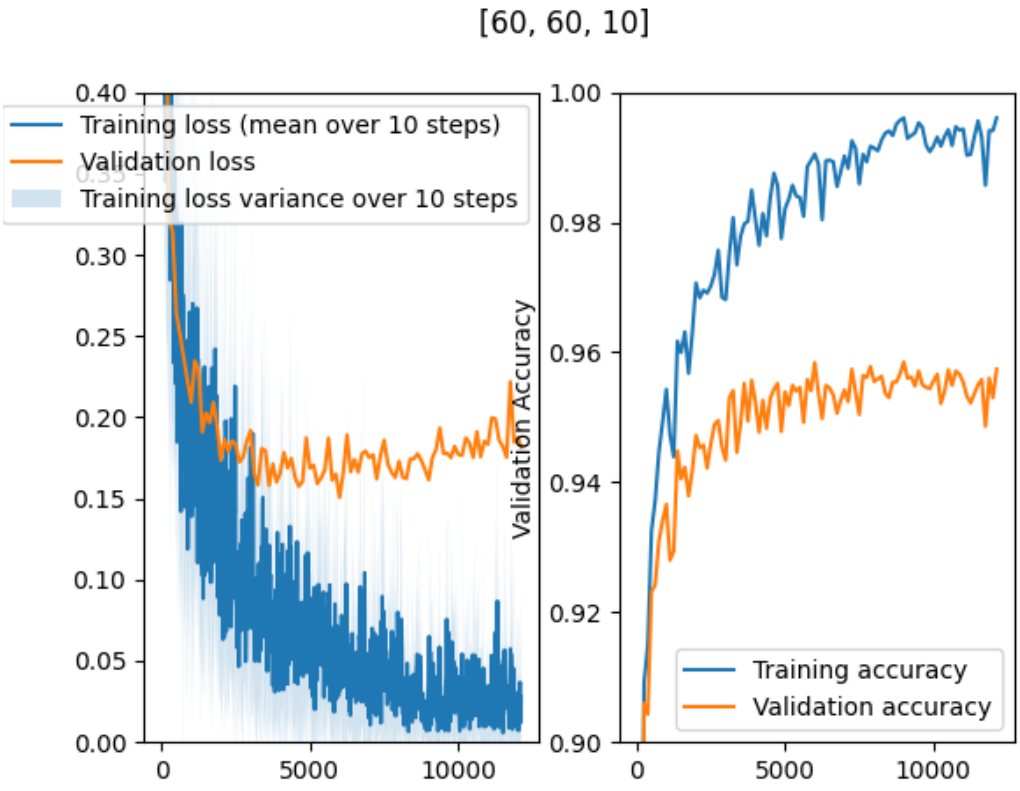


5.3 Task 4d)

There are $(28 \cdot 28 + 1) \cdot 64 + (64 + 1) \cdot 10 = 50890$ parameters in with nodes per layer $[784, 64, 10]$

N = number of nodes in each hidden layer General number of parameters with two hidden layers of size N: $(28 \cdot 28 + 1) \cdot N + (N + 1) \cdot N + (N + 1) \cdot 10 = N^2 + 796N + 10$

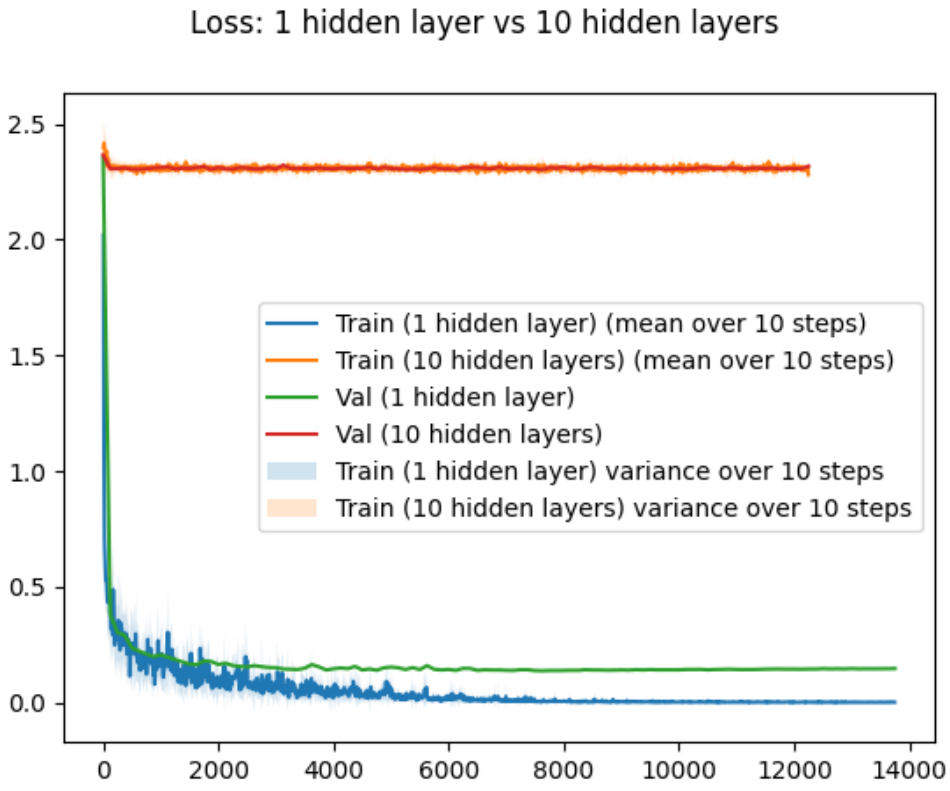
Solving $N^2 + 796N + 10 = 50890 \implies N \approx 59.476$. Choosing $N = 60$ results in $60^2 + 796 * 60 + 10 = 51370$ parameters.

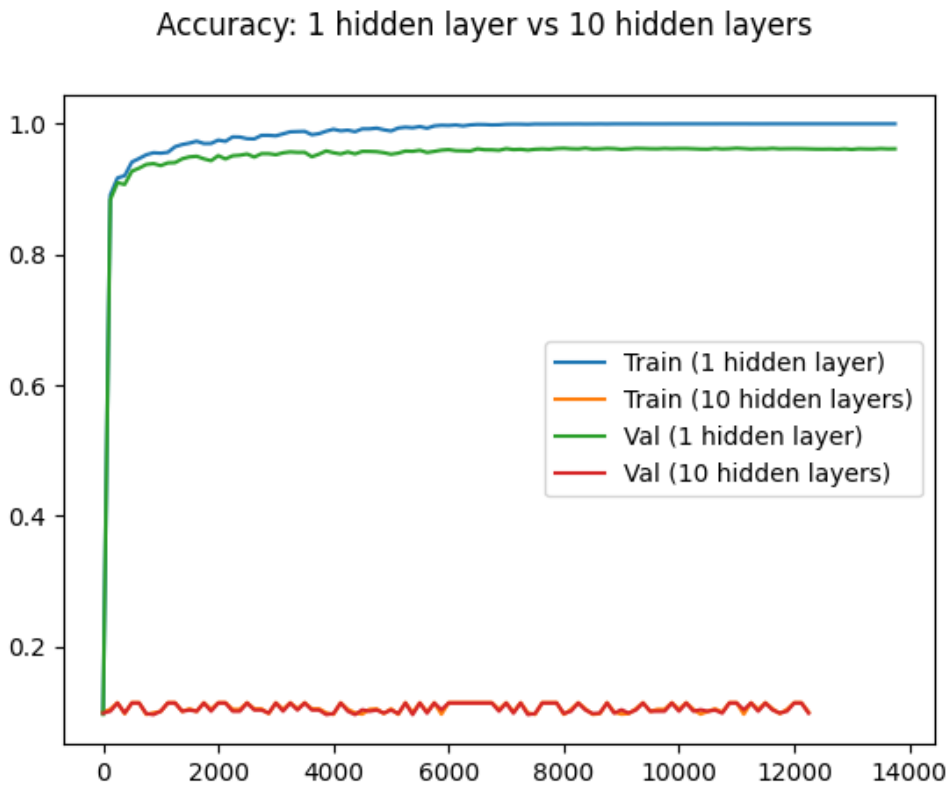


With two hidden layers containing 60 nodes each, we see the accuracy is more noisy than before. It is also a bit lower than in the previous network. The loss is also more noisy towards the end, whereas the in task 3 (with both improved weight initialization and momentum) it stops oscillating at the end.

However, it looks like there is less overfitting with this network. This is assumed from looking at how the training accuracy does not continue to improve much after the validation accuracy seem to converge.

5.4 Task 4e)


Loss: 1 hidden layer vs 10 hidden layers
2.5
2.0
1.5
1.0
0.5
0.0
0
2000
4000
6000
8000
10000
12000
14000
Train (1 hidden layer) (mean over 10 steps)
Train (10 hidden layers) (mean over 10 steps)
Val (1 hidden layer)
Val (10 hidden layers)
Train (1 hidden layer) variance over 10 steps
Train (10 hidden layers) variance over 10 steps



From the plots, it looks like the network with 10 hidden layers never improves. A reason for this could be that it's too deep… The information from the output layer used in calculating the derivatives become less and less useful the further back we propagate.

[ ]: