Assignment 2 Report

Group 16

Hanna Waage Hjelmeland, Siri Holde Hegsvold

Task 1

task 1a)

Output:

$$w_{kj} := w_{kj} - \alpha \frac{\partial C}{\partial w_{kj}} = w_{kj} - \alpha \delta_k a_j \quad (1)$$

where $\delta_k = \frac{\partial C}{\partial z_k}$

Hidden:

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}} \quad (2)$$

can use $\delta_j = \frac{\partial C}{\partial z_j}$ (*)

Task 1a: Backpropagation

By using the definition of δ_j, show that (2) can be written as:

$$w_{ji} := w_{ji} - \alpha \delta_j x_i$$

and that $\delta_j = f'(z_j) \Sigma_k w_{kj} \delta_k$

First try to rewrite $\alpha \frac{\partial C}{\partial w_{ji}}$ using the chain rule

$$\alpha \frac{\partial C}{\partial w_{ij}} = \alpha \delta_j a_i = \alpha \frac{\partial C}{\partial z_j} x_i$$

using (*) & $a_i = x_i$

$$= \alpha \left(\sum_k \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial z_j} \right) x_i$$

$$= \alpha \left(\sum_k \delta_k \frac{\partial z_k}{\partial z_j} \right) x_i$$

$z_k = \sum_j^{J} w_{kj} \cdot x_j \quad (x_i = a_j)$ (from last assignment)

$$= \alpha \left(\Sigma_k \delta_k \left(\frac{\partial \Sigma_j w_{kj} \cdot a_j}{\partial z_j} \right) \right) x_i$$

$$= \alpha \left(\Sigma_k \delta_k \left(\frac{\partial \Sigma_j w_{kj} \cdot f(z_j)}{\partial z_j} \right) \right) x_i$$

$a_j = f(z_j)$ from information about notation in intro

$$= \alpha \left(\Sigma_k \delta_k w_{kj} f'(z_j) \right) x_i$$

$$= \alpha \left(f'(z_j) \Sigma_k \delta_k w_{kj} \right) x_i$$

$= \alpha \delta_j x_i$ where δ_j must be $f'(z_j) \Sigma_k \delta_k w_{kj}$

[redacted]

task 1b)

Task 1b: Vectorize computation

Show update rule for the weight matrix form

1. Hidden layer to output layer

Normal:

$$w_{kj} = w_{kj} - \alpha \frac{\partial C}{\partial w_{kj}} = w_{kj} - \alpha \delta_k a_j$$

Vectorizing:

$$\mathbf{W}_{kj} = \mathbf{W}_{kj} - \alpha \boldsymbol{\delta}_k \mathbf{a}_j^T$$

$\mathbf{W}_{kj}$.shape $= (64, 10)$

$\boldsymbol{\delta}_k$.shape $= (100, 10)$

$\mathbf{a}_j$.shape $= (100, 64)$

2. Input layer to hidden layer

Normal:

$$w_{ji} = w_{ji} - \alpha \underbrace{f'(z_j) \sum_k w_{kj} \delta_k}_{\delta_j} \cdot x_i$$

Vectorizing:

$$\boldsymbol{\delta}_j = \mathbf{f}'(\mathbf{z}_j) \circ \boldsymbol{\Sigma}_j$$

$$\mathbf{W}_{ji} = \mathbf{W}_{ji} - \alpha \mathbf{f}'(\mathbf{z}_j) \circ \boldsymbol{\Sigma}_j \, \mathbf{x}_i^T$$

$$= \mathbf{W}_{ji} - \alpha \boldsymbol{\delta}_j \mathbf{x}_i^T$$

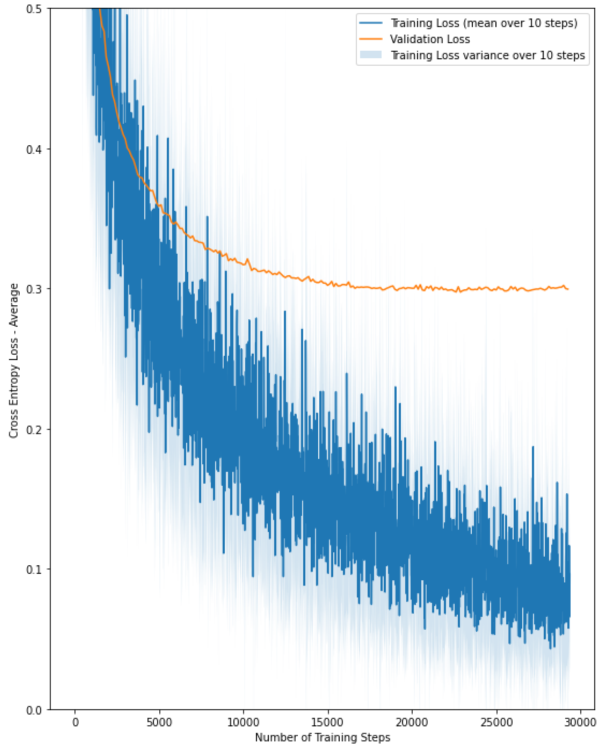
$\mathbf{W}_{ji}$.shape $= (785, 64)$

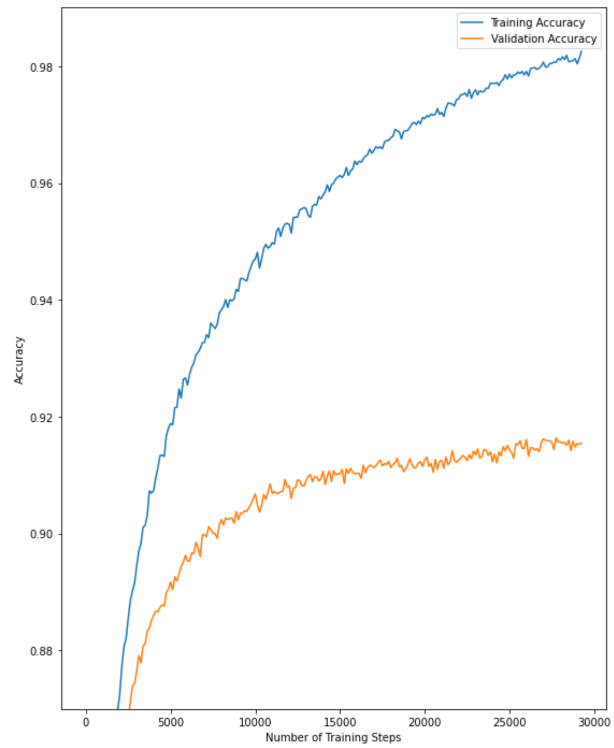
$\boldsymbol{\delta}_j$.shape $= (100, 64)$

$\mathbf{x}_i$.shape $= (100, 785)$

Task 2 c)

```
Final Train Cross Entropy Loss: 0.07770593272819233
Final Validation Cross Entropy Loss: 0.2994116833858604
Train accuracy: 0.9823
Validation accuracy: 0.9164
```





Comment: The network works, but *very* slowly. We also see signs of overfitting (as the training accuracy continues to increase while the validation accuracy stagnates).

Task 2d)

Number of parameters = number of weights + number of biases The number of weights is the number of connections between the nodes of the input, hidden and outputlayer. The number of biases is the number of nodes in the hidden and outputlayer.

w = i * j + j * k

b = j + k

sum = i*j + j*k + j + k = 784*64 + 64*10 + 64 + 10 = 50890

Task 3:

The tricks are added incrementally.

We make 3 comparisons:

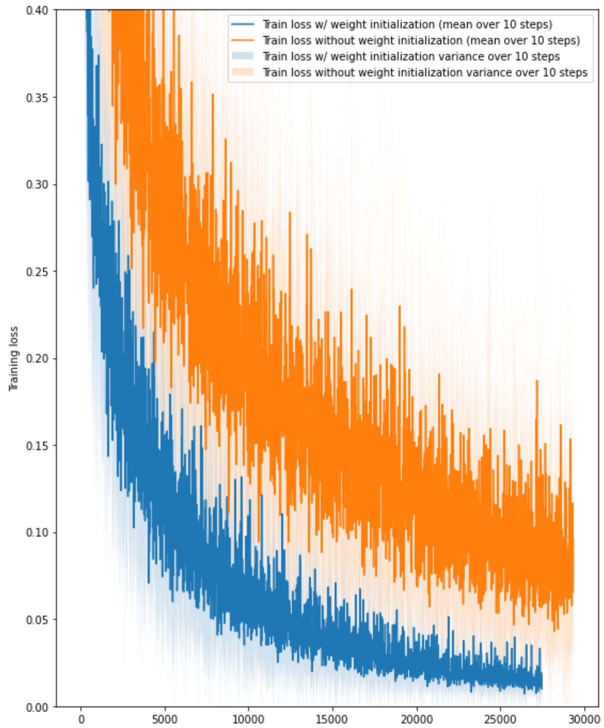
```
# With and without:
# 1. weight init
# 2. improved sigmoid
# 3. momentum
```

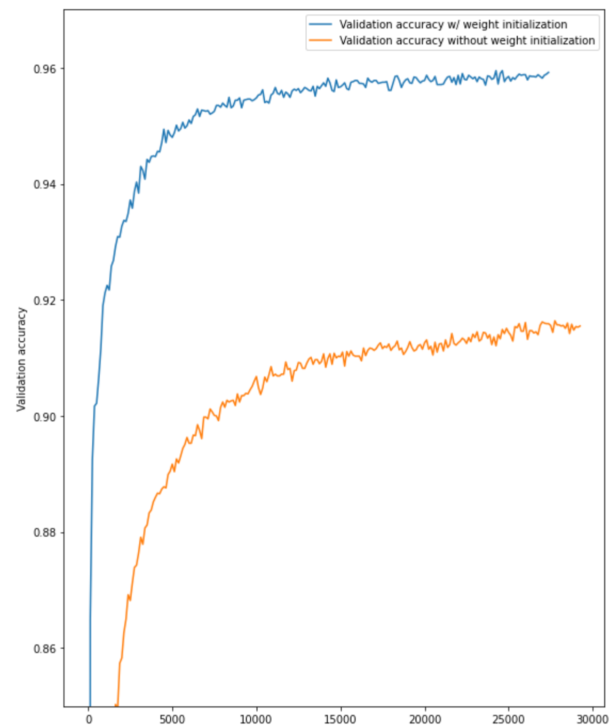
PS: The trick from the previous step is implemented in the next, e. g. when adding the improved sigmoid, we have the weight initialization trick implemented.

Plots of comparisons:

3a) Implementing weight initialization from normal distribution

```
Final Train Cross Entropy Loss for initial model: 0.07770593272819233
Final Validation Cross Entropy Loss for initial model: 0.2994116833858604
Train accuracy for initial model: 0.9823
Validation accuracy for initial model: 0.9164
Final Train Cross Entropy Loss for model with weight initialization: 0.013895749
Final Validation Cross Entropy Loss for model with weight initialization: 0.1426
Train accuracy for model with weight initialization: 0.99955
Validation accuracy for model with weight initialization: 0.958
```



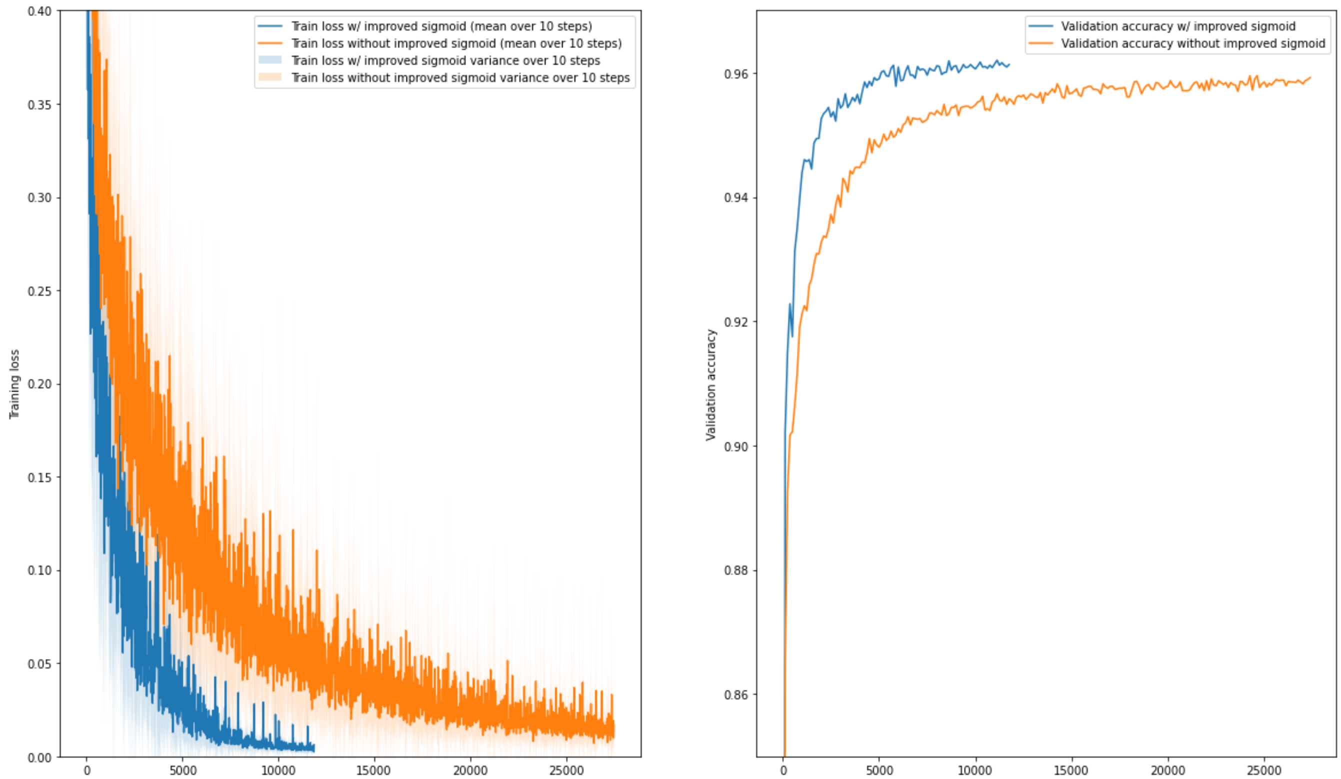


Comparison (with weight initialization as opposed to without):

- Convergence speed is faster (training loss decreases more rapidly)
- Better generalization (validation accuracy is much better, all the way from the start)
- Final accuracy and loss has improved both for the training and the validation set.

3b) Implementing the improved sigmoid

```
Final Train Cross Entropy Loss for model with weight initialization: 0.013895749
Final Validation Cross Entropy Loss for model with weight initialization: 0.1426
Train accuracy for model with weight initialization: 0.99955
Validation accuracy for model with weight initialization: 0.958
Final Train Cross Entropy Loss for model with improved sigmoid: 0.00387384100639
Final Validation Cross Entropy Loss for model with improved sigmoid: 0.148627663
Train accuracy for model with improved sigmoid: 0.99995
Validation accuracy for model with improved sigmoid: 0.9617
```

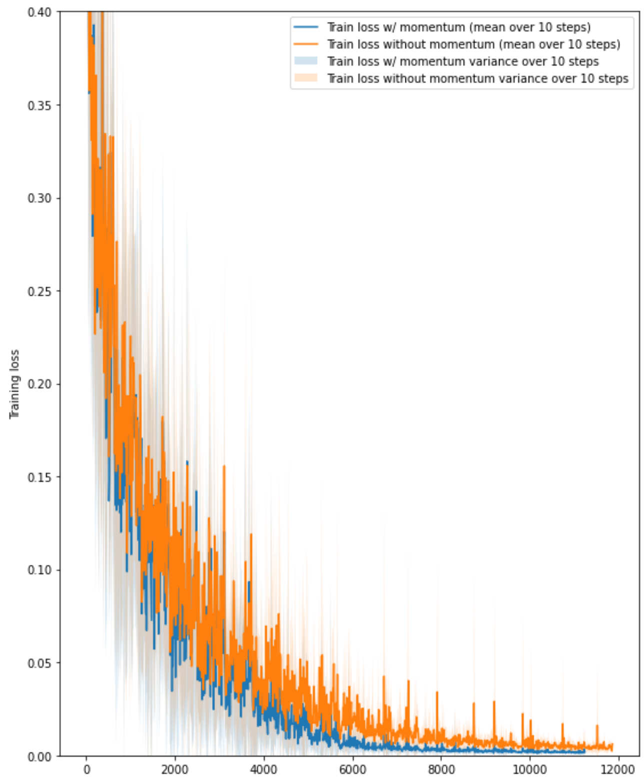


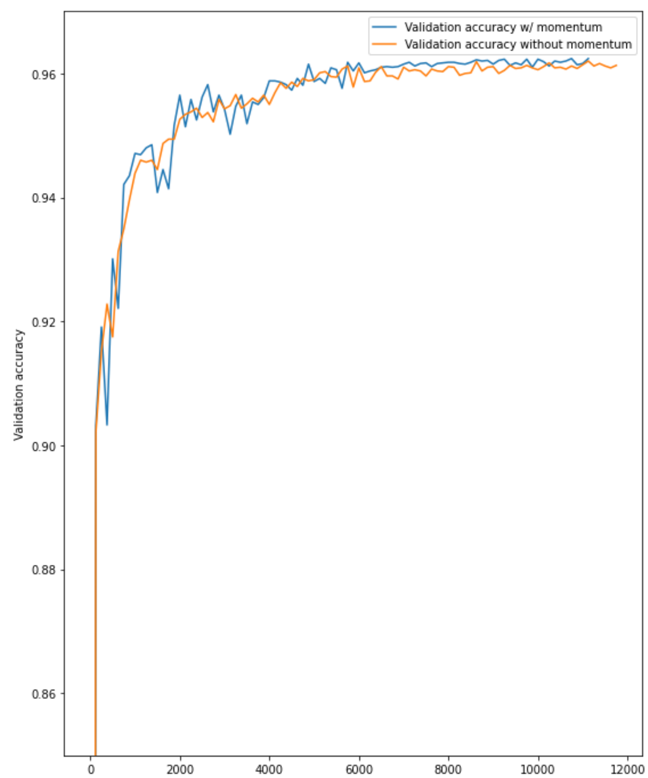
Comparison (with improved sigmoid as opposed to without):

- Convergence speed is faster (training loss decreases more rapid)
- Better generalization (validation accuracy is better). We also see that the training stops early, which can prevent overfitting.
- Final accuracy and loss stays very similar to the previous, except for training loss which has a more significant decrease. Validation loss increases a little bit.

3c) Implementing momentum

```
Final Train Cross Entropy Loss for model with improved sigmoid: 0.00387384100639
Final Validation Cross Entropy Loss for model with improved sigmoid: 0.148627663
Train accuracy for model with improved sigmoid: 0.99995
Validation accuracy for model with improved sigmoid: 0.9617
Final Train Cross Entropy Loss for model with momentum: 0.001605220786994835
Final Validation Cross Entropy Loss for model with momentum: 0.1612057320623876
Train accuracy for model with momentum: 1.0
Validation accuracy for model with imomentum: 0.9619
```





Comparison (with momentum as opposed to without):

- Convergence speed is somewhat faster
- Generalization seems to be at the same level as before. (The fact that the model with improved sigmoid did not stop early here is likely due to a different seed in random for the weight-initalization).

- Final accuracy and loss stays very similar to the previous.

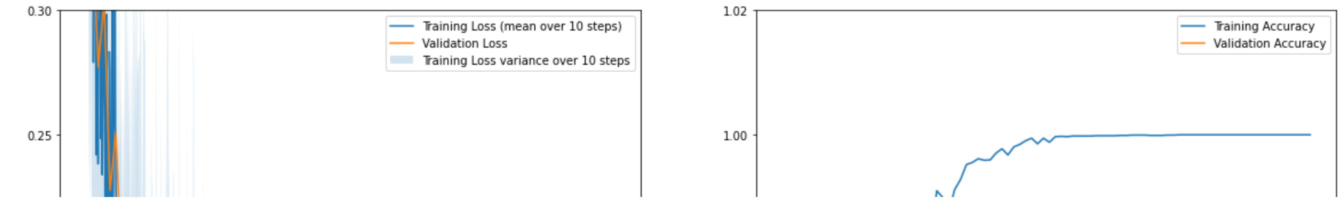
Conclusion: it seems wise to implement all of the trics of the trade.

Task 4

For reference: Result from baseline from task 3

- all tricks implemented
- one hidden layer of 64 neurons

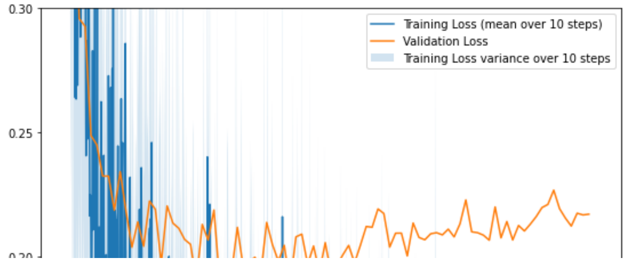
```
Final Train Cross Entropy Loss: 0.0016052207869949714
Final Validation Cross Entropy Loss: 0.16120573206239205
Train accuracy: 1.0
Validation accuracy: 0.9619
```

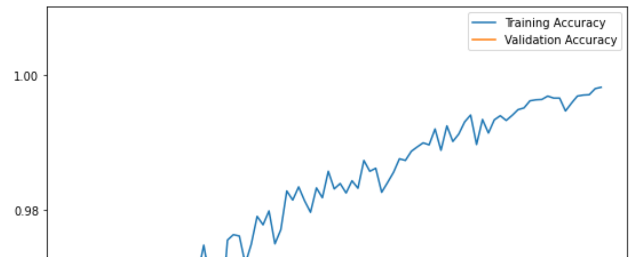


Task 4a)

Result with 32 hidden units:

```
Final Train Cross Entropy Loss: 0.012859409030197 79
Final Validation Cross Entropy Loss: 0.21462587387440277
Train accuracy: 0.99805
Validation accuracy: 0.9493
```



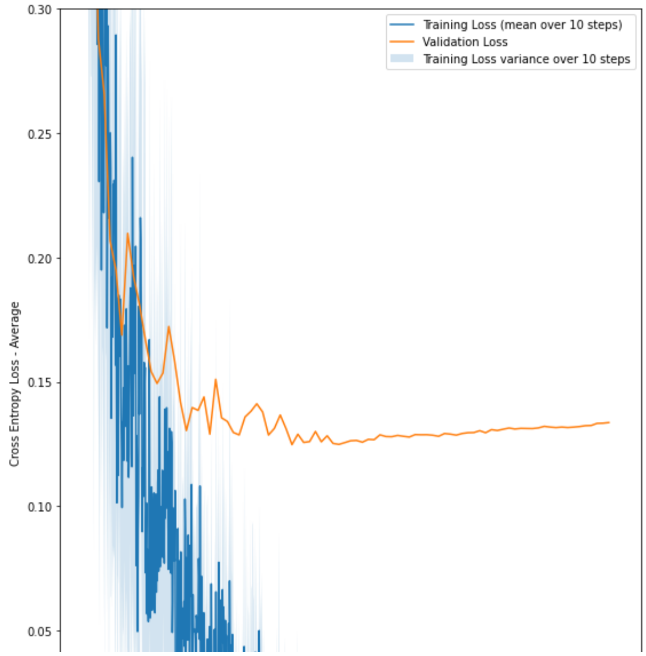


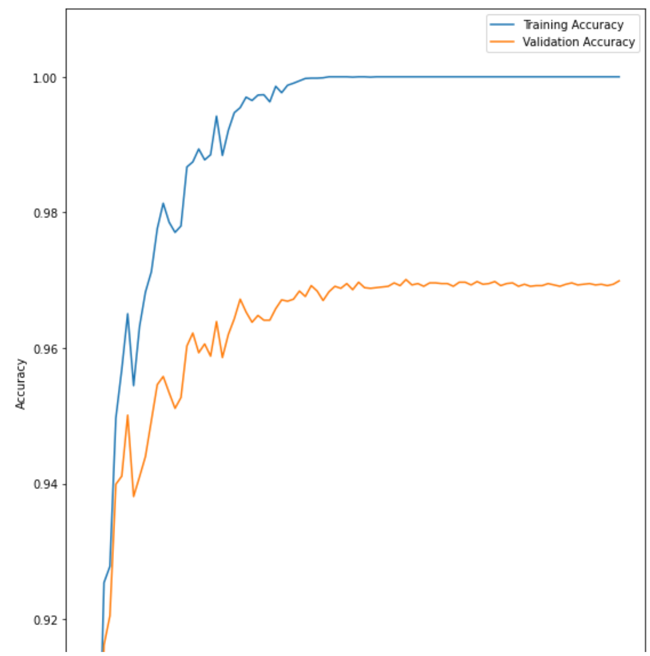
We see that the resulting performance is slightly worse (but code takes less time to run). The convergence is also slower. We also see that the the accuracies and losses are a bit more noisy, which can be due to every neuron cathing less nuanced features due to the smaller number of neurons. Thus, every neuron has to cover a broader "responsibility" which means that it, at every training step, more likely will change its value more dramatically than with more neurons.

Task 4b)

Result with 128 hidden units:

```
Train shape: X: (20000, 784), Y: (20000, 1)
Validation shape: X: (10000, 784), Y: (10000, 1)
Initializing weight to shape: (785, 128)
Initializing weight to shape: (128, 10)
Early stopping at epoch:  17
Final Train Cross Entropy Loss: 0.0008191917453922721
Final Validation Cross Entropy Loss: 0.1328643114507488
Train accuracy: 1.0
Validation accuracy: 0.9693
```





We see that increasing the number of parameters in the hidden layer improves convergence time, reduces the loss and somewhat improves the validation accuracy. When the number of hidden units become too high, we introduce more complexity to the model which increases the risk of overfitting. Also, a very large number of neurons in the hidden layers can increase the time it takes to train the network. In the extreme case the amount of training time can increase to the point where it is impossible to adequately train the neural network. That is not a problem in this case, but the time it took to train the network did increased a little.

Task 4d) Two hidden layers

Let x be the number of nodes in the hidden layers. Then

$w = i * x + x * x + x * k$

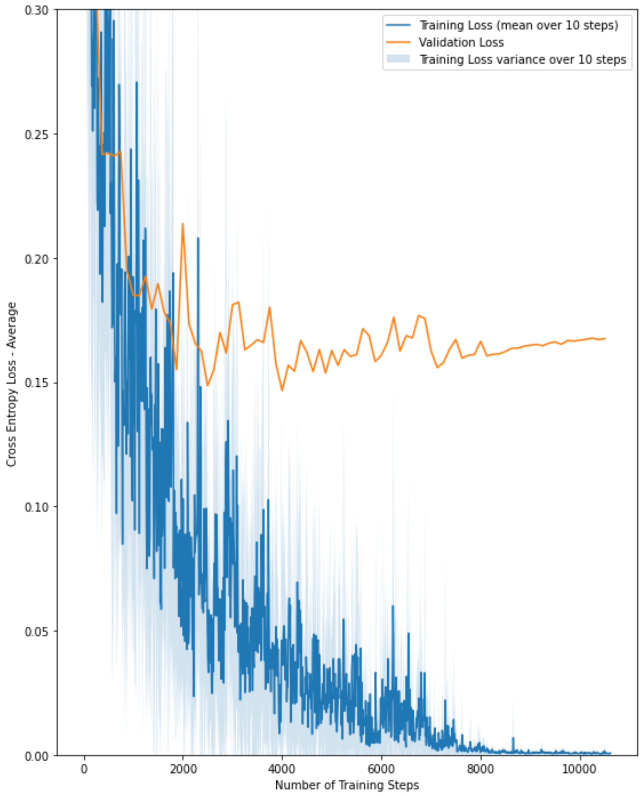
$b = 2*x + k$

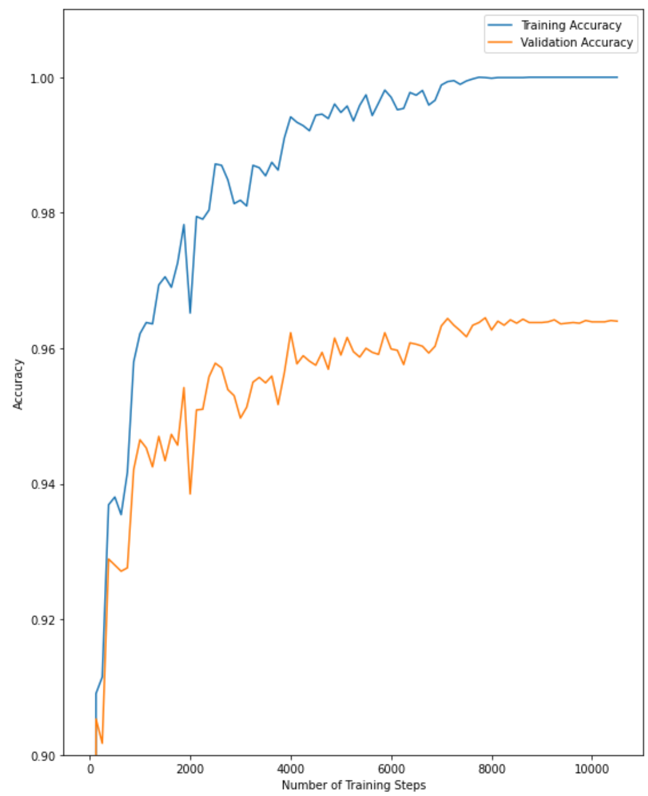
Want approximately the same number of parameters as in task 3, i. e. 50890 (task 3 has the same number of parameters as calculated in 2d). This yields an equation which we choose to solve in Geogebra, yielding that the closest power of 2 to solve the equation is x = 64.

1	k := 10 → $k := 10$
2	i := 784 → $i := 784$
3	w(x):= i*x + x*x + x*k → $w(x) := x^2 + 794\,x$
4	b(x):= 2*x + k → $b(x) := 2\,x + 10$
5	S(x):=w+b → $S(x) := x^2 + 796\,x + 10$
6	S = 50890 Løs: $\left\{x = -2\sqrt{52321} - 398, x = 2\sqrt{52321} - 398\right\}$
7	{x = -2 sqrt(52321) - 398, x = 2sqrt(52321) - 398} ≈ $\{x = -855.48, x = 59.48\}$
8	S(64) ≈ 55050

Thus, we use 64 nodes in the hidden layers. This gives 55050 parameters.

```
Train shape: X: (20000, 784), Y: (20000, 1)
Validation shape: X: (10000, 784), Y: (10000, 1)
Initializing weight to shape: (785, 64)
Initializing weight to shape: (64, 64)
Initializing weight to shape: (64, 10)
Early stopping at epoch:  16
Final Train Cross Entropy Loss: 0.0005760154222621541
Final Validation Cross Entropy Loss: 0.1682561304705854
Train accuracy: 1.0
Validation accuracy: 0.9642
```





The performance seems a lot like the performance of the model in the previous step (with 128 nodes) except it converges a little slower.

Task 4e)

Assuming "baseline model from task 3" means the model with all tricks implemented, and with only one hidden layer with 64 neurons:

```
Downloading train-images-idx3-ubyte.gz...
Downloading t10k-images-idx3-ubyte.gz...
Downloading train-labels-idx1-ubyte.gz...
Downloading t10k-labels-idx1-ubyte.gz...
(47040000,)
(7840000,)
(60000,)
(10000,)
Train shape: X: (20000, 784), Y: (20000, 1)
```

We see worse performance when increasing to 10 layers. This is likely due to the way backpropagation works - when the architecture is very deep, the gradient can vanish when propagating and thus the training becomes very ineffective.

```
Initializing weight to shape: (64, 10)
Early stopping at epoch:  25
Final Train Cross Entropy Loss with 10 layers: 0.06464538550135918
Final Validation Cross Entropy Loss with 10 layers: 0.2128677091691643
Train accuracy with 10 layers: 0.98095
Validation accuracy with 10 layers: 0.9477
Initializing weight to shape: (785, 64)
Initializing weight to shape: (64, 10)
Early stopping at epoch:  17
```