1a

For the weights of the hidden layer, we have that the gradient descent rule with learning rate α is

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}$$

Now, we want to show that this may be rewritten in terms of other variables. Firstly, we make some definitions. Let $\delta_k = \frac{\partial C}{\partial z_k} = -(y_k - \hat{y_k})$, where k denotes an index in the output layer. We try to rewrite the gradient. The last equality is known from the previous assignment.

Since the loss C is affected by every neuron in the output layer, we sum over the contributions:

$$\frac{\partial C}{\partial w_{ji}} = \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial w_{ji}}$$

We expand the last term. Since z_k only affected by w_{ji} through node j in the hidden layer, we have that:

$$\frac{\partial C}{\partial w_{ji}} = \sum_{k=1}^{K} \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}}$$

We recognize that $\frac{\partial C}{\partial z_k} = \delta_k$, $\frac{\partial z_k}{\partial a_j} = w_{kj}$ (the weight of the activation in the hidden layer node j towards the input of node k in the output layer), $\frac{\partial a_j}{\partial z_j} = f'(z_j)$ and that $\frac{\partial z_j}{\partial w_{ji}} = x_i$. Thus:

$$\begin{aligned}\frac{\partial C}{\partial w_{ji}} &= \sum_{k=1}^{K} \delta_k w_{kj} f'(z_j) x_i \\ &= (f'(z_j) \sum_{k=1}^{K} \delta_k w_{kj}) x_i\end{aligned}$$

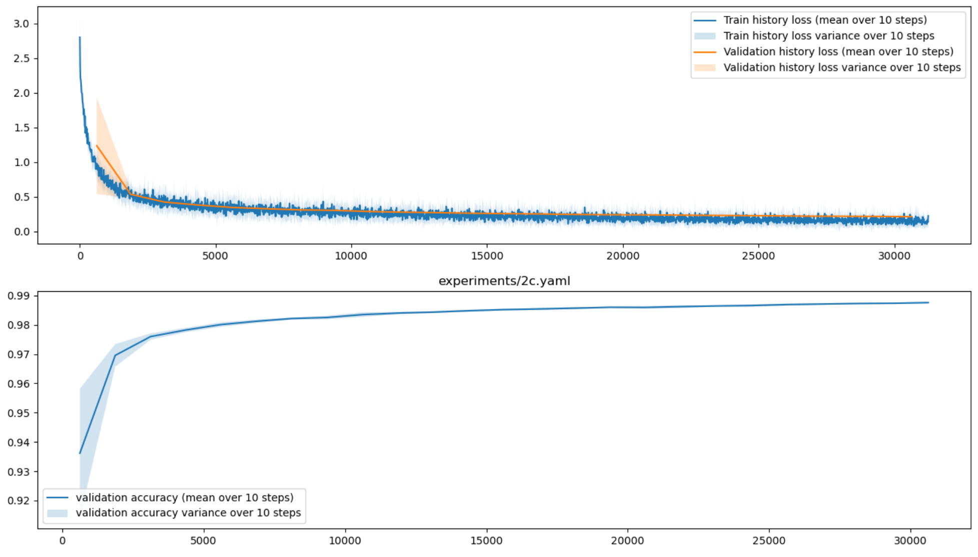
Defining $\delta_j = f'(z_j) \sum_{k=1}^{K} \delta_k w_{kj}$, we have that:

$$\frac{\partial C}{\partial w_{ji}} = \delta_j x_i$$

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

Task2

We trained the network with one hidden layer of 64 neurons, with random initialization of weights.



The network achieves a validation accuracy of $\approx 0.98781\%$

The network has parameters according to

$$n_{parameters} = (n_{inputs} + 1) \cdot n_{hidden} + n_{hidden} \cdot n_{out}$$

Where the term "+1" comes from the bias trick. In our case, this equals

$$n_{parameters} = (784 + 1) \cdot 64 + 64 \cdot 10 = 50880$$

Our network has 50880 parameters. This may also be verified through counting the number of elements in the arrays in ‘model.ws‘ in the code.

Task3

We see that with each trick added, the training improves. Note that early stopping is activated so that 3b and 3c are done much faster than 3a.

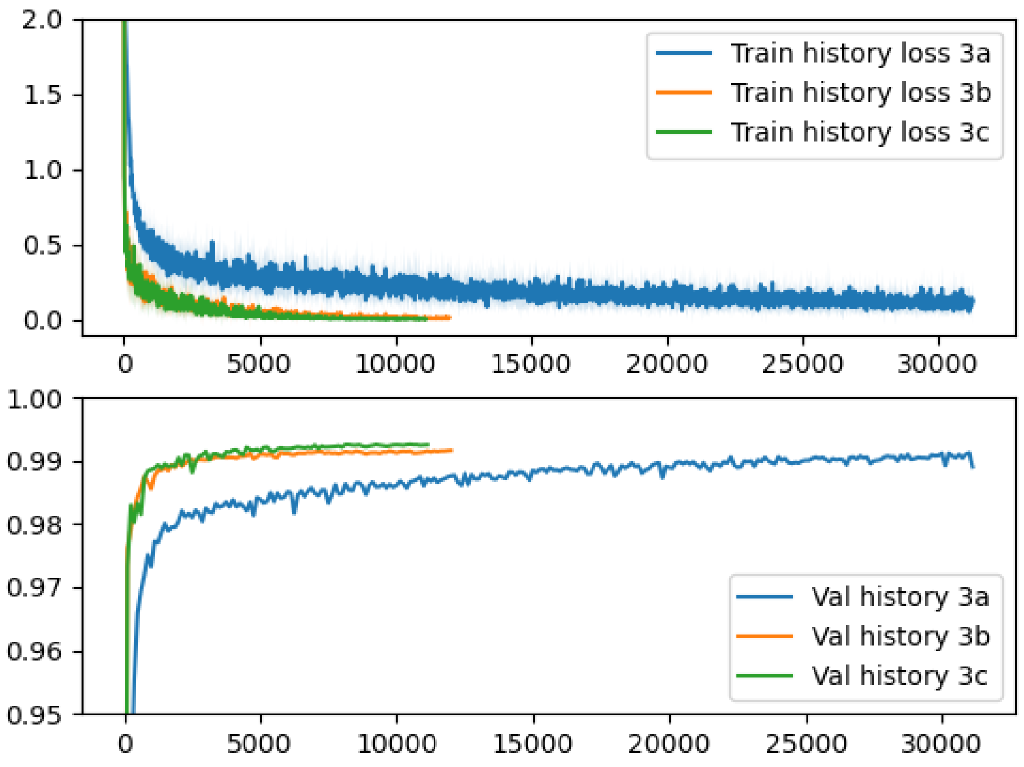


Figure 1: Improvements when adding tricks

Task4

Comparing the three network structures with different hidden neuron counts: 32, 64 and 128 was interesting. Clearly having more neurons give a higher accuracy. The biggest difference was computation time - training took slightly longer for the larger nets due to calculating forward and backward are slightly more costly with the bigger matricies.

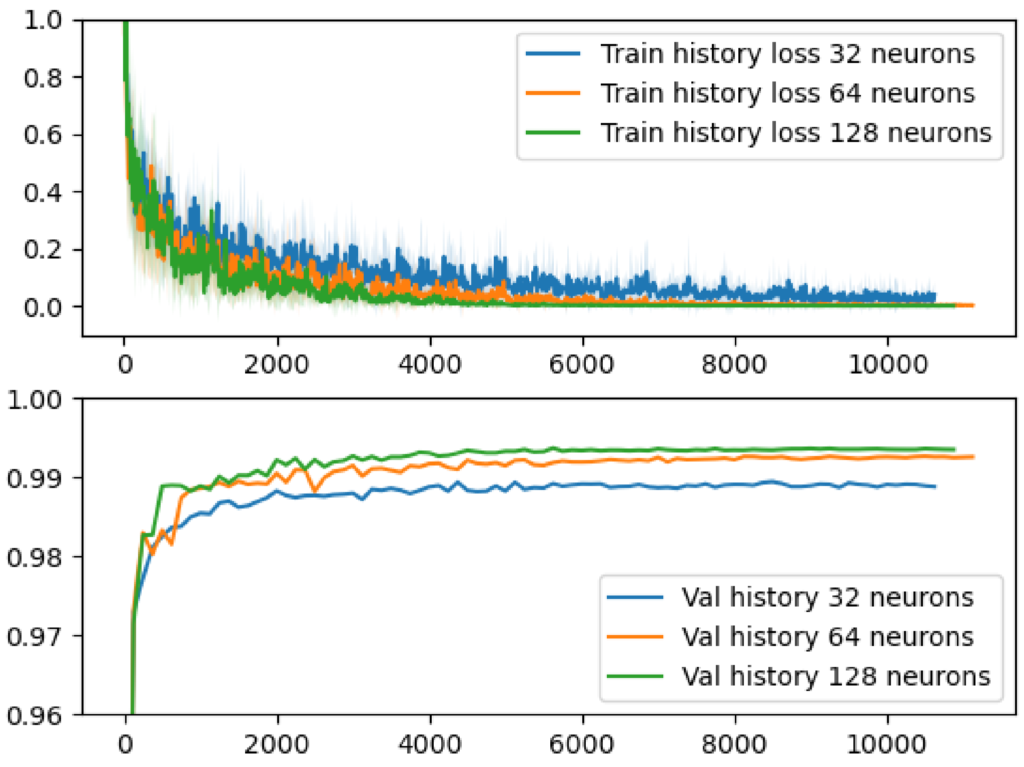


Figure 2: Differences with network structure

Comparing two network structures with the same number of parameters, but a different depth was not so interesting.

Using 59 hidden units in both layers, we get approximately the same amount of parameters. In the previous layer we had 50880 parameters, while here we have:

$$n_{parameters} = (784 + 1) \cdot 59 + 59 \cdot 59 + 59 \cdot 10 = 50386 \approx 50880$$

They seem to behave very similarly, producing the same accuracy. No difference was observed in training.

This makes sense, as there are basically the same number of parameters to tune. The biggest difference is that the deeper network gets a bit more nonlinearity-opportunity due to the additional layer of sigmoid neurons. This did not have any noticable effect though.

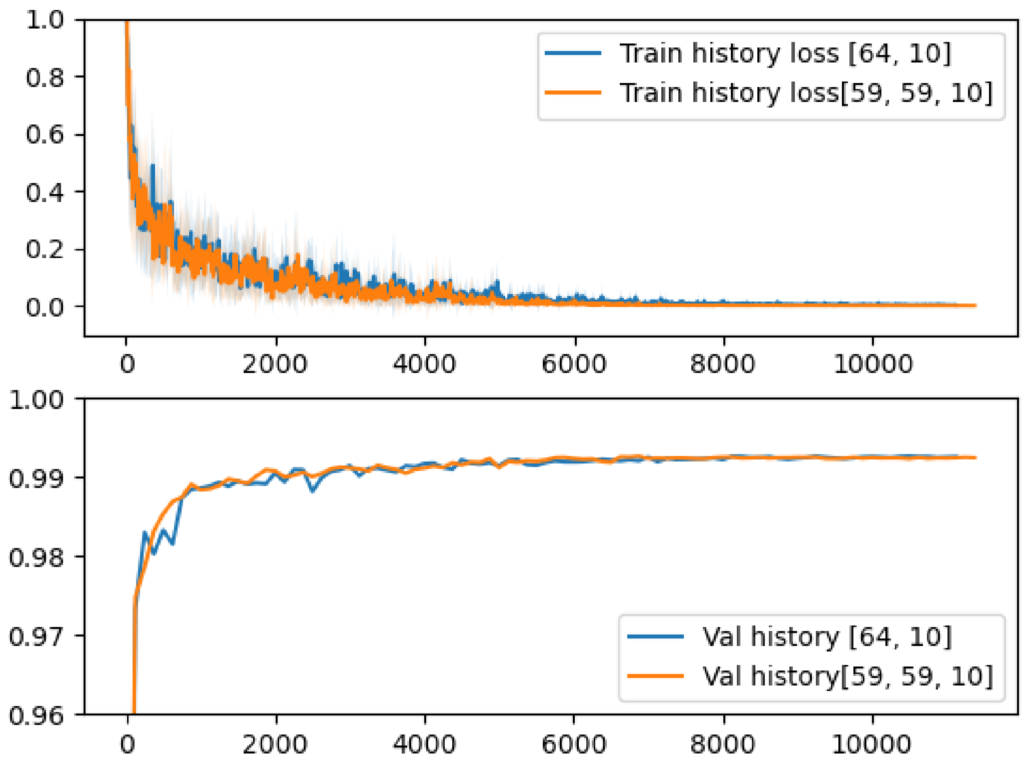


Figure 3: Differences with one or two hidden layers

Training a deep network with 10 hidden layers took some time (numerical from more iterations and more weights), but it did not yield any big differenes in performance. The two nets are very similar.

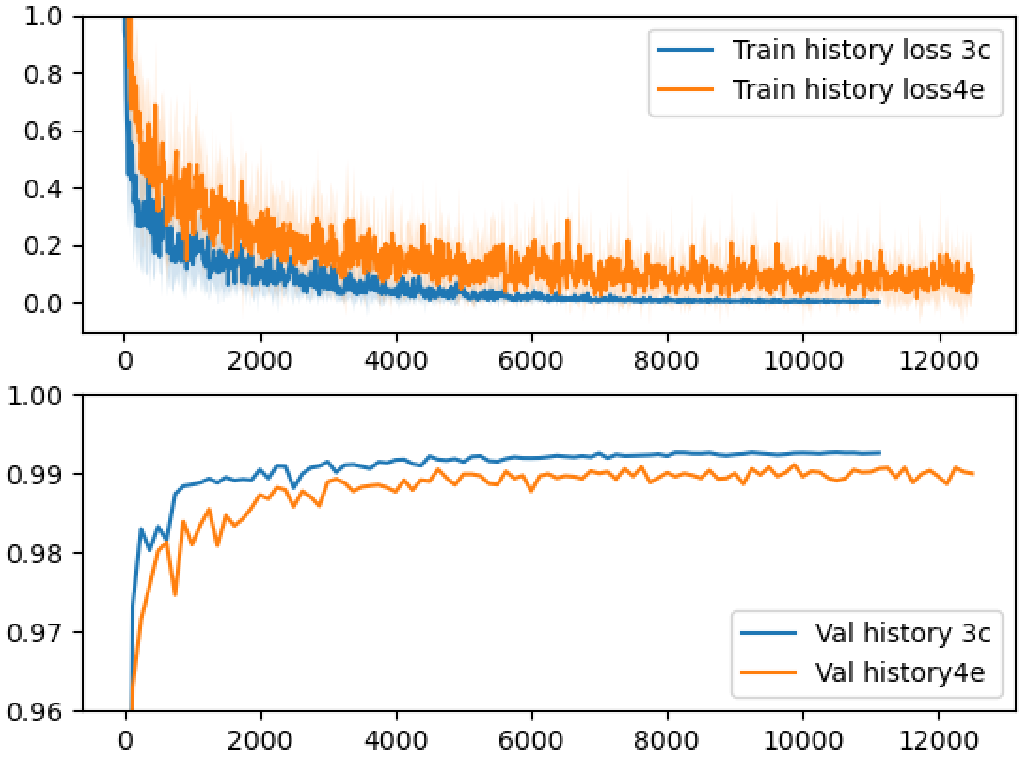


Figure 4: Differences with deep vs shallow network