TDT4265 - Computer Vision and Deep Learning

Assignment 2

Martin Johannes Nilsen

17th February 2022

1 Softmax regression with backpropagation

1.1 Task 1a

Given in the assignment as equation 2 we have that

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

The chain rule gives us

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}}$$

The next thing we can do is to substitute

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

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

Further on we can calculate the δ_j

$$\delta_j = \frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} = \frac{\partial C}{\partial a_j} \cdot f'(z_j)$$

We have left to find $\frac{\partial C}{\partial a_j}$

$$\frac{\partial C}{\partial a_j} = \left(\sum_k \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial a_j} \right)$$

Based on this we can simplify as we know that

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

$$\frac{\partial z_k}{\partial a_j} = \frac{\partial}{\partial a_j} \left(\sum_j w_{kj} a_j \right) = w_{kj}$$

If we put it together we may end up with the wished equation

$$\delta_j = \frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial a_j} f'(z_k) = f'(z_k) \cdot \sum_k w_{kj} \delta_k$$

1.2 Task 1b

Hidden → output layer:

We have that

$$W^K := \alpha \cdot a_j \cdot {\delta_k}^T$$

where the dimensions of $W^K = (J \times K)$. This can be confimed by the following equation:

$$\dim(a_j) \times \dim({\delta_k}^T) = (J \times 1) \times (1 \times K) = (J \times K)$$

Input → hidden layer:

We have that

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

which we converts to matrix notation as follows:

$$W_j := W_j - \alpha x_i {\delta_j}^T$$

We also know that $\delta_j = f'(z_j) \sum_k w_{kj} \delta_k$, which we can convert into matrix form

$$\delta_j = f'(z_j) \circ (W_k \cdot \delta_K)$$

and find the dimensions

$$\begin{aligned}
\dim(\delta_j) &= \dim(z_j) \circ \dim(W_k \cdot \delta_K) \\
&= \dim\left(W_j^T \cdot x_i\right) \circ \dim(W_k \cdot \delta_K) \\
&= (J \times I) \times (I \times 1) \circ (J \times K) \times (K \times 1) \\
&= (J \times 1) \circ (J \times 1) \\
&= (J \times 1)
\end{aligned}$$

Which gives us the final sanity test for the dimension of W_j

$$\begin{aligned}
\dim(W_j) &= \dim(x_i) \times \dim\left(\delta_j^T\right) \\
&= (I \times 1) \times (1 \times J) \\
&= (I \times J)
\end{aligned}$$

2 Softmax Regression with Backpropagation

2.1 Task 2a

The reported mean and standard deviation are 33.55 and 78.87.

2.2 Task 2c

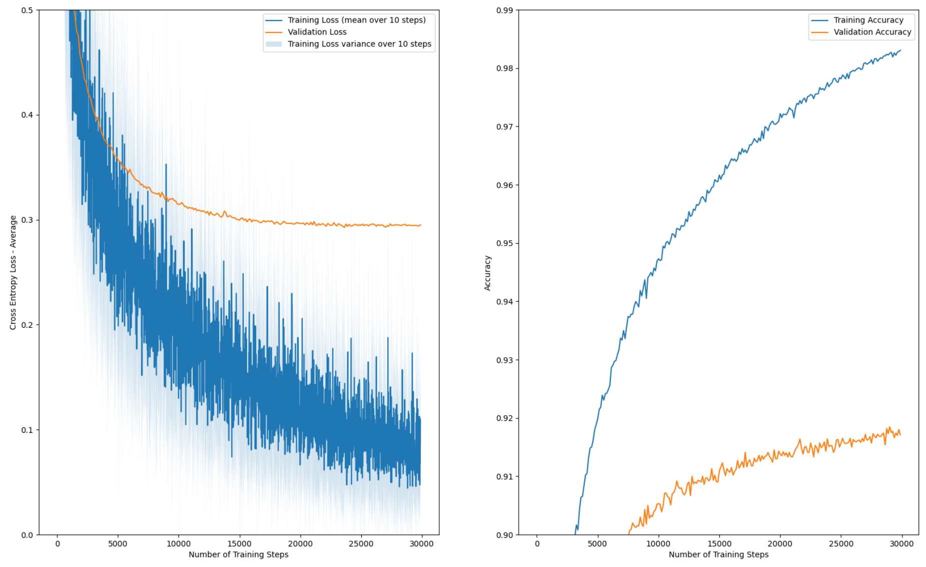


Figure 1: Loss and accuracy over training and validation. Keep in mind that I have implemented early stopping from the last assignment. The number of epochs needed to improve before stopping is increased to 50, from 10 in assignment 1.

2.3 Task 2d

The parameter count is $(28 * 28 + 1) * 64 + 64 * 10 = 50880$. The first paranthesis covering the weights in addition to one bias (bias trick), before we take the hidden units and outputs into account.

3 Adding the "Tricks of the Trade"

We were tasked to implement the three different *tricks of the trade*; improved sigmoid, improved weight initialization and momentum. In the source code provided there was inteded to plot the models with and without data shuffling. When I ran the code it was not too much of an observable difference to spot, but as it is generally known to be the better choice, I decided to keep it in the following plots.

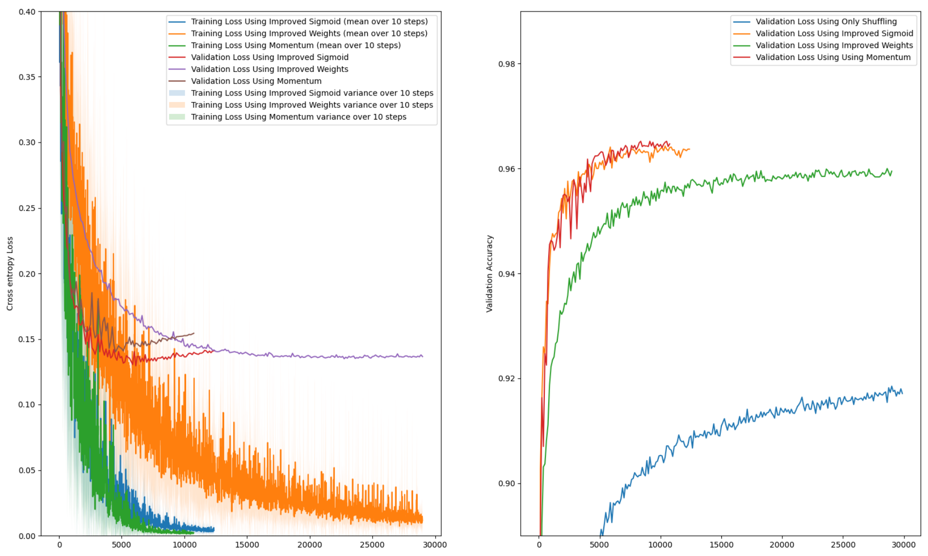


Figure 2: Tricks of the trade

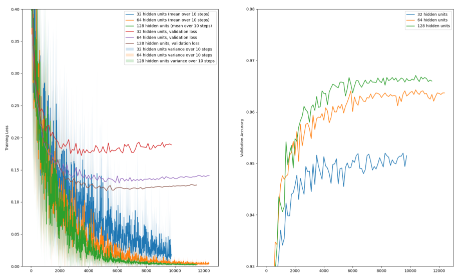
In Figure 3 the different tricks of the trade are implemented, in addition to the use of early stopping. The first observation one may do is regarding the validation accuracy. The addition of improved sigmoid is one of the best additions regarding accuracy and speed of convergence. The addition of improved weight initialization seems to give the best effect on training loss, making the validation accuracy more stable but generally lower than with the usage of momentum or improved sigmoid alone. Momentum seems to be making it converge even faster, but it seems like the model has a slight problem with overfitting as the validation loss is beginning to increase even with early stopping. Regarding learning speed we see that each addition seem to have some effect here, with improved sigmoid being the strongest!

When it comes to convergence the effect of improved weight initialization seems to have the largest effect. This makes the early stopping not kick in, and even though the validation accuracy is not the highest, it is the most stable and seems to converge a little below 96%. It should be noted here that it seems to be a problem with the generalization in the models. One may observe signs of overfitting when looking at the validation loss against the training loss. The improved weights seems to be fixing this problem, slightly punishing the accuracy but generalizing the best. The other loss curves can be observed having quite a steep increasing tendency before being stopped by early stopping.

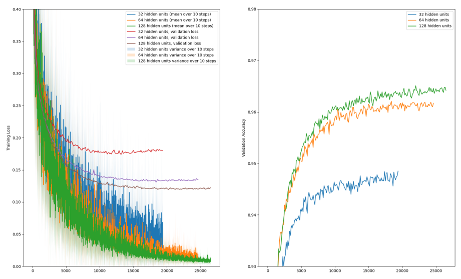
4 Experiment with network topology

4.1 Task 4a and b

What will happen if we either decrease or increase the amount of hidden units? This is what we where tasked to find out in task a and b. As a base I used the model from the previous task, but as I was a little bit uncertain about the usage of momentum, I have plotted the figures both with and without momentum. In the plotting below, we can observe that the validation loss and accuracy seems to be more stable using momentum.



(a) Tricks of the trade without momentum

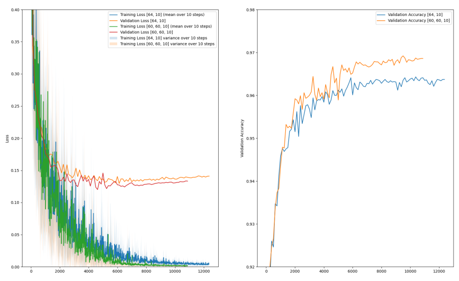


(b) Tricks of the trade with momentum

Figure 3: Tricks of the trade

From Figure 3 we can observe that more hidden units yield better accuracy. However, it should be noted that it is more computationally expensive to use more hidden units. When it comes to validation, we can see that having too few neurons yields overfitting (increasing loss) with the lack of a good accuracy, and having too many neurons causes the network to take longer to stop learning. So to summarize, the increas of hidden units seems to yield better training and accuracy, on the behalf of a longer training period and an increased computational expense.

4.2 Task 4d


(a) Without momentum

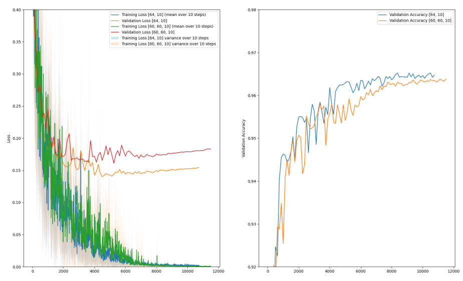
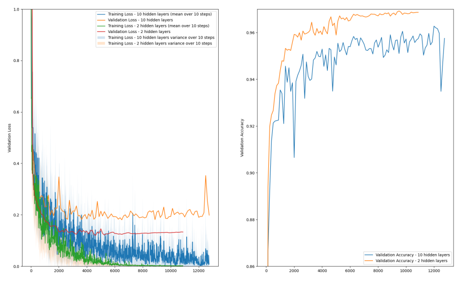

(b) With momentum

Figure 4: Network with 2 hidden layers (60,60,10)

In this task we were to train a network consisting of two hidden layers, having approximately the same amount of parameters as the model from task 3. As we had 50880 parameters in task 3, I ended up on the dimension of $(60, 60, 10)$, which yields $(28 * 28 + 1) * 60 + 60 * 60 + 60 * 10 = 51300$ parameters. If you take a closer look at the accuracy in these plots against the accuracy of the model from the former task with 64 hidden layer neurons (units), there seems to be a slight increase in the accuracy without momentum. With momentum, it seems to perform a little bit worse than the former model.

4.3 Task 4e


(a) Without momentum

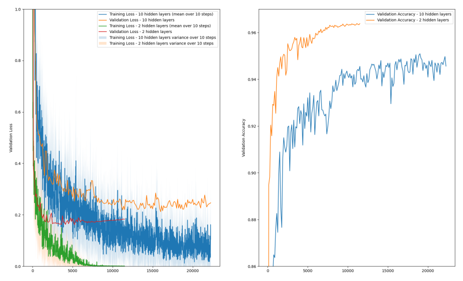

(b) With momentum

Figure 5: Network with 10 hidden layers (64,...,10)

We were tasked to increase the amount of hidden layers to ten layers of 64, with the last layer being of size 10. As we know from the regularization in assignment 1, the complexity of the model can punish the learning of the model. This can bee seen in Figure 5 as the accuracy-plots above show that the model with 10 hidden layers seems to not be able to achieve the same accuracy as the one with 2 hidden layers. The number of parameters with 10 hidden layers is $(785) * 64 + (64 * 64) * 9 + 64 * 10 = 87744$, which is about a 70% increase. This seem to affect the validation accuracy, in addition to increasing the magnitude of the noise/oscillations.