TDT4265 Assignment 2

Anders Thallaug Fagerli

Task 1

a)

The update rule for the hidden layer is

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}, \tag{1}$$

where $\frac{\partial C}{\partial w_{ji}}$ can be rewritten using the chain rule as

$$\frac{\partial C}{\partial w_{ji}} = \sum_k \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}}.$$

The individual terms are found as

$$\begin{aligned}
\frac{\partial C}{\partial z_k} &= \delta_k, \\
\frac{\partial z_k}{\partial a_j} &= \frac{\partial}{\partial a_j} \left[\sum_{j=0}^{J} w_{kj} a_j \right] = w_{kj}, \\
\frac{\partial a_j}{\partial z_j} &= \frac{\partial}{\partial z_j} f(z_j) = f'(z_j), \\
\frac{\partial z_j}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \left[\sum_{i=0}^{I} w_{ji} x_i \right] = x_i,
\end{aligned}$$

and by inserting back into (1), we get

$$w_{ji} := w_{ji} - \alpha f'(z_j) \left(\sum_k w_{kj} \delta_k \right) x_i.$$

By defining $\delta_j = f'(z_j) \sum_k w_{kj} \delta_k$, the update rule is finally written as

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

b)

Assuming we have N data samples and K classes, we can rewrite δ_j into matrix form as

$$\boldsymbol{\delta}_j = \boldsymbol{f}'(\mathbf{Z}_j) \odot \boldsymbol{\delta}_k \mathbf{W}_{kj}^{\top},$$

where $\odot$ is the Hadamard product, giving the update rule in matrix form as

$$\mathbf{W}_{ji} := \mathbf{W}_{ji} - \alpha \mathbf{X}^{\top} \boldsymbol{\delta}_j,$$

where the dimensions of the matrices are given as

$$\mathbf{Z}_j : (N \times J), \qquad \boldsymbol{\delta}_k : (N \times K),$$
$$\mathbf{W}_{kj} : (J \times K), \quad \mathbf{W}_{ji} : (I \times J),$$
$$\boldsymbol{\delta}_j : (N \times J), \qquad \mathbf{X} : (N \times I).$$

For the update rule from the hidden layer to the output layer, we have the gradient

$$\begin{aligned}\frac{\partial C}{\partial w_{kj}} &= \frac{\partial C}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}} \\ &= \delta_k a_j,\end{aligned}$$

which gives the update rule in matrix form as

$$\mathbf{W}_{kj} := \mathbf{W}_{kj} - \alpha \mathbf{A}_j^\top \boldsymbol{\delta}_k,$$

where $\mathbf{A}_j$ is of size $(N \times J)$.

Task 2

c)

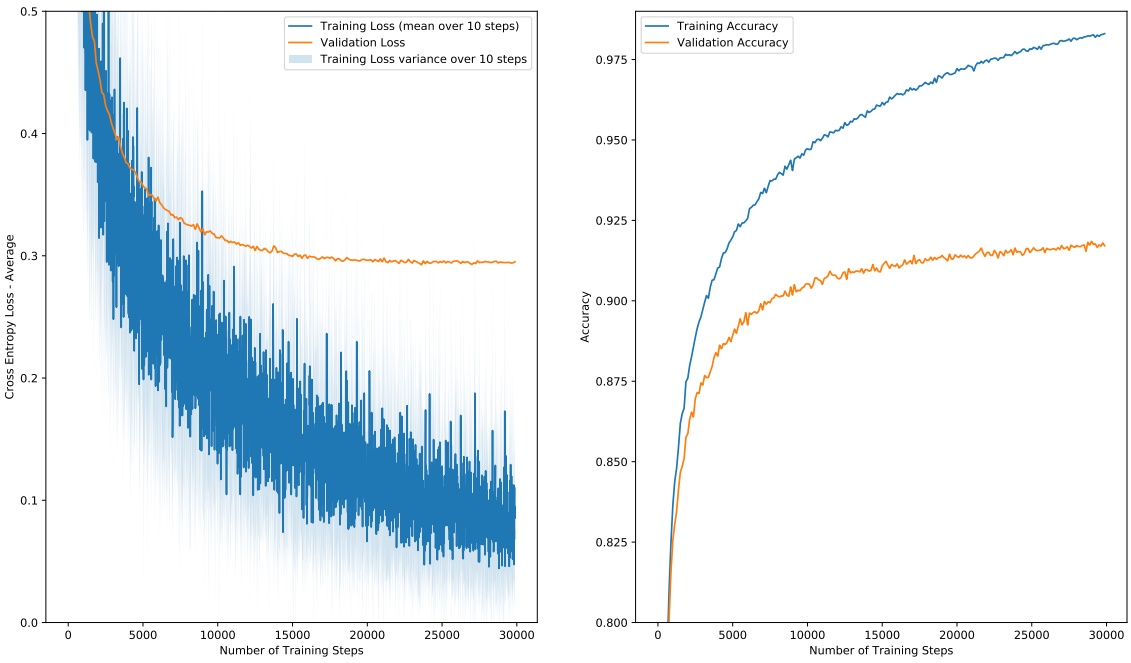


Figure 1: *Training and validation set loss (left) and accuracy (right) over training.*

d)

We have 785 input nodes, 64 hidden nodes and 10 output nodes, giving a total of $785 \times 64 + 64 \times 10 = 50880$ parameters in the network.

Task 3

a), b) and c)

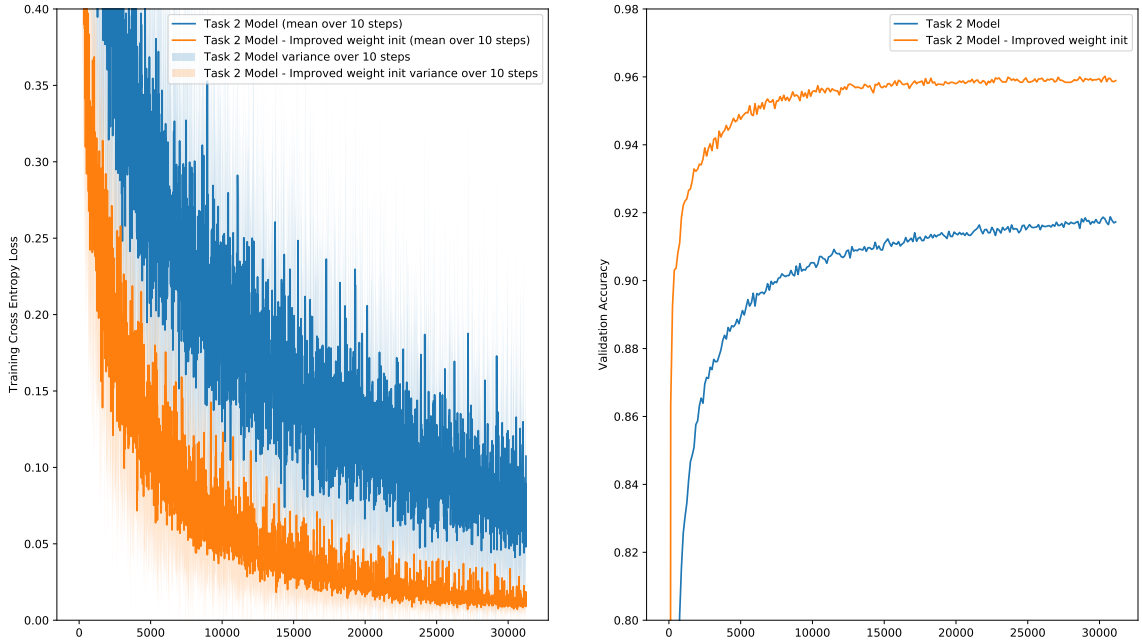


Figure 2: *Training loss (left) and validation accuracy (right) with and without improved weight initialization.*

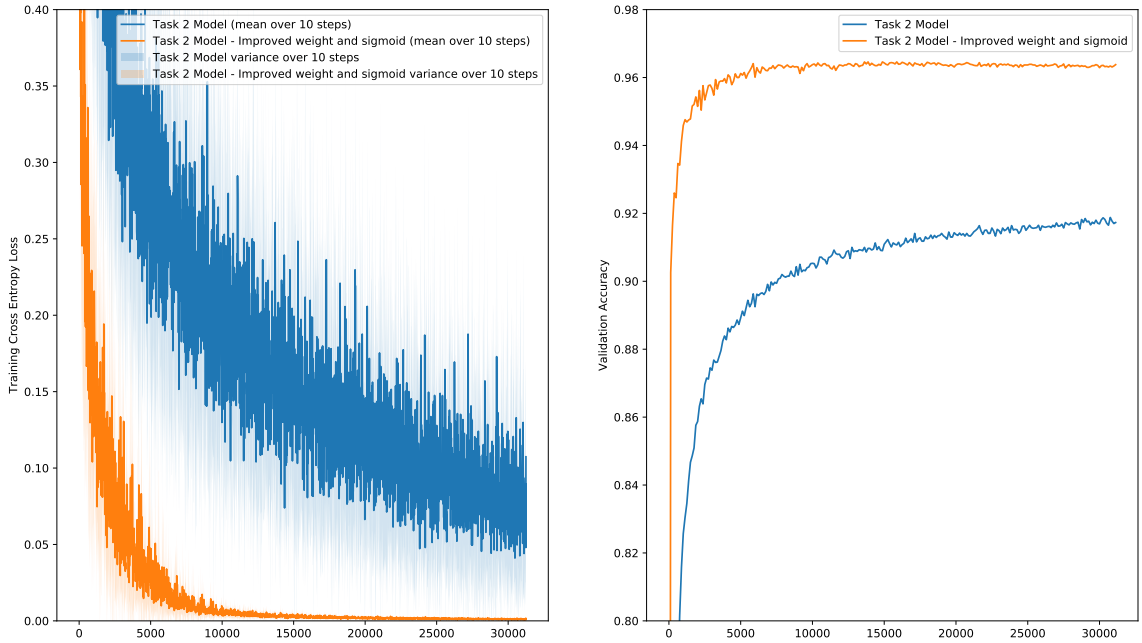


Figure 3: *Training loss (left) and validation accuracy (right) with and without improved sigmoid and weight initialization.*

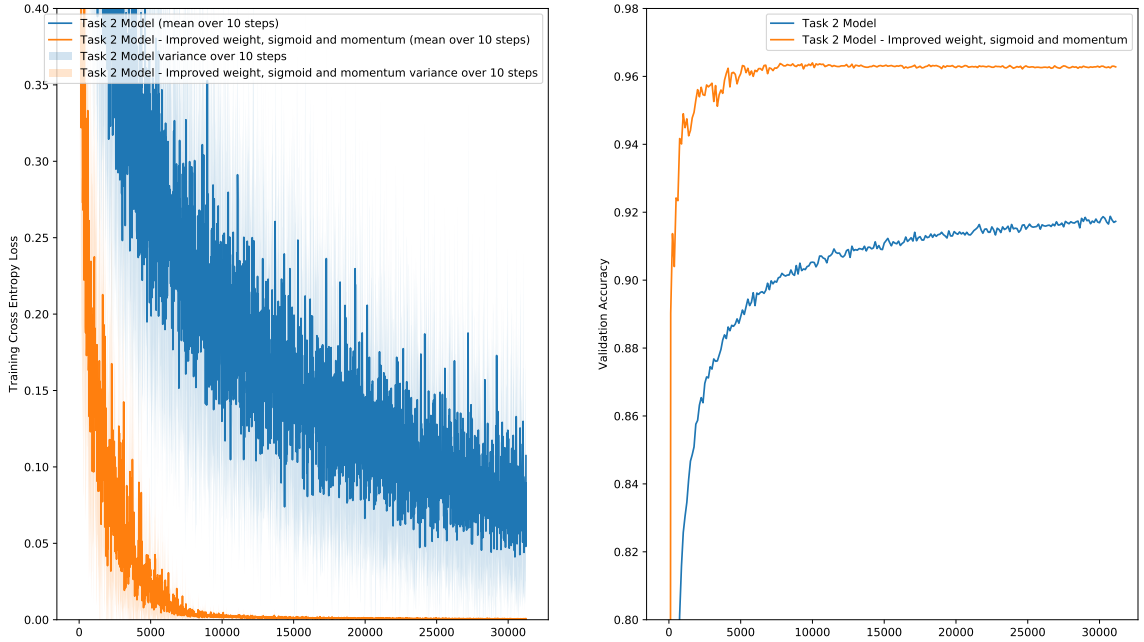


Figure 4: *Training loss (left) and validation accuracy (right) with and without momentum, improved sigmoid and weight initialization.*

In Figure 2 we see an improvement in both convergence speed and accuracy by introducing an improved weight initialization. This is because the weights now are zero-centered, so the weight update can drive the individual weights in either direction. The sigmoid is however not zero-centered, meaning the input to the hidden layer will be biased in some direction, which again decreases the convergence speed.

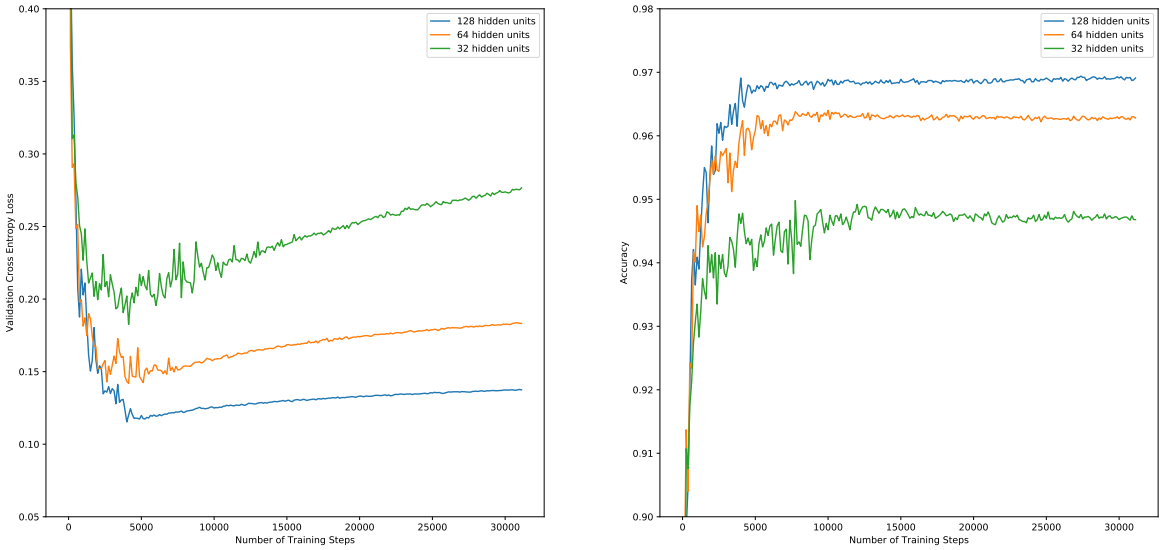
By using the improved sigmoid, the convergence speed is much better, as seen in Figure 3. The same argument as above applies, and since the output of each layer now is zero-centered, the weight update for all layers will not be biased in any direction. The weight initialization together with the improved sigmoid also ensures that the variance is close to 1 over the layers, such that the sigmoid is active over a larger area. With too large weights the sigmoid will saturate, giving too small gradients, and with too small weights the sigmoid will only be active in the linear area, so nonlinearities will not be modeled. This heavily affects the convergence speed, which again affects the accuracy when we use a maximum number of iterations as termination criteria.

Momentum is finally added in Figure 4, but is not seen to impact the convergence speed or accuracy to any large extent. A slight increase in convergence speed is seen from the training loss, but by further inspection we also find that the accuracy has slightly decreased. This is probably just that the gradient descent has converged to a differnet minima due to the momentum, but it is insignificant in terms of overall performance.

As the training loss approaches zero when adding these tricks, one should be aware of possible overfitting. From Figure 1 we see clear signs of overfitting, as the training loss and accuracy is much better than for the validation set. The tricks we have implemented do not increase generalization in any way, so the problem of overfitting remains for the network in Task 3.

Task 4

a) and b)



(a) Loss with varying number of hidden units. *(b) Accuracy with varying number of hidden units.*

Figure 5: *Validation set loss and accuracy over training with varying number of hidden units.*

In Figure 5 we see the results of using 128, 64 and 32 units in our hidden layer for the validation set. It seems that using too few units leads to slower convergence and poorer accuracy, meaning it might not be enough to capture the complexity of the underlying model. It is however much faster computationally. The network with 128 units performs quite well, giving the highest accuracy, but is also computationally most demanding by a large margin. The small increase in accuracy from the network with 64 units should therefore be weighed against the increase in computation time when choosing network topology.

d)

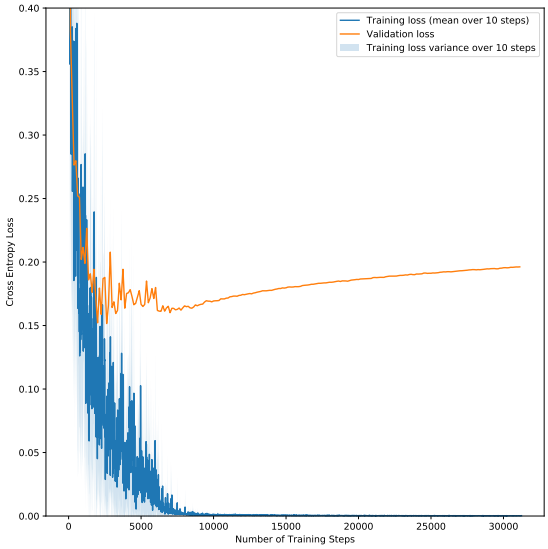
From Task 3 we had 50880 parameters. To achieve around the same number of parameters with two hidden layers, we use 60 units in each hidden layer, so that the total number of parameters is $785 \times 60 + 60 \times 60 + 60 \times 10 = 51300$.

From Figure 6 we see that the network performs similarly to the one in Task 3, although the training takes much more time. The previous network is therefore preferred for this model. Overfitting is now also apparent from the plot of training and validation loss, where the training loss goes to zero while the validation loss increases after some time.

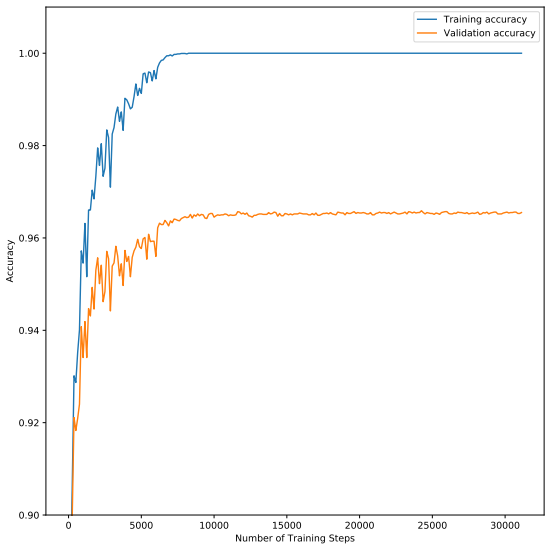
e)

In Figure 7 we see the comparison between a network with ten hidden layers of 64 neurons each, and a network with two hidden layers of 60 neurons each (same as baseline from Task 3). There are now $785 \times 64 + 9 \times 64 \times 64 + 64 \times 10 = 87744$ parameters in the network, so the network has a much higher complexity than before. This means a small change in input may give a large change in output, and we see from Figure 7 that the network zig-zags much more because of this, giving a much slower convergence. This is analogous to fitting a high order polynomial to a few points from an unknown model, which will

heavily miss a new point. In the same way, the network overfits a batch while training, and when a new batch is given the network struggles to adapt to the new input.

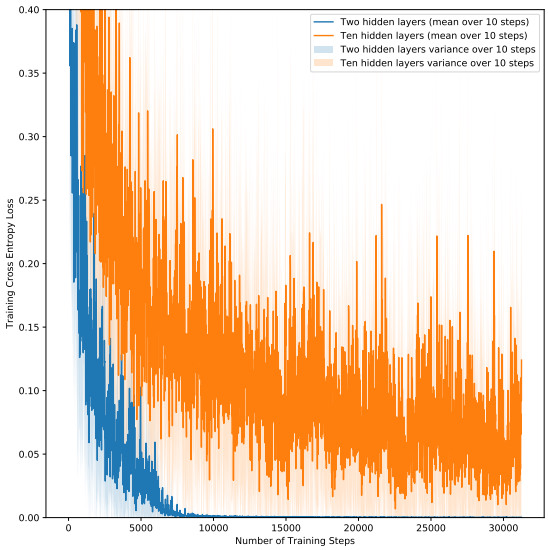


(a) Training and validation loss.

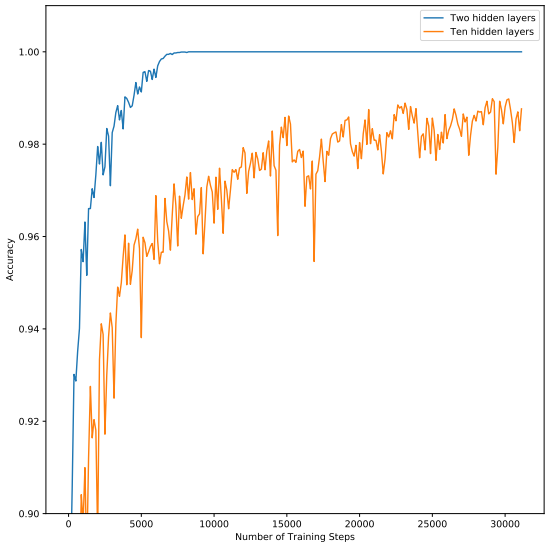


(b) Training and validation accuracy.

Figure 6: *Cross entropy loss and accuracy with two hidden layers of 60 units each.*



(a) Training loss.



(b) Training accuracy.

Figure 7: *Cross entropy loss and accuracy on training set for two different network topologies.*