TDT4265 - Computer Vision & Deep Learning

Assignment 2 Report - Group 66

> Dionysios Rigatos
> dionysir@stud.ntnu.no

NOTE: Assignment01 was delivered by me while part of Group 113. I am now part of Group 66.

Task 1

The learning rate α will be thus defined as η for this task, as a personal preference.

Task 1a)

We want to show that $w_{ji} := w_{ji} - \eta \frac{\partial C}{\partial w_{ji}}$ is equivalent to $w_{ji} := w_{ji} - \eta \delta_j x_i$.

Given $\delta_j = \frac{\partial C}{\partial z_j}$, we will show that $\frac{\partial C}{\partial w_{ji}} = \delta_j x_i$.

The assumption that the bias term is included in the weights is made.

Using the chain rule, we can write:

- $\delta_j = \frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial a_j} \frac{\partial a_j}{\partial z_j} (1)$

Also, we can expand:

- $\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}} (2)$

And given (1), we can write:

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

where we can expand...

- $\frac{\partial z_j}{\partial w_{ji}} = \frac{\partial (w_{ji} x_i)}{\partial w_{ji}} = x_i (3)$

And thus, from (1) and (3) we have that (2) becomes:

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

Finally, we can write:

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

$\square$

For the second part;

We want to show that $\delta_j = f'(z_j) \sum_k w_{kj} \delta_k$, given $\delta_k = \frac{\partial C}{\partial z_k}$.

We start by re-writing our equation as $\delta_j = \frac{\partial C}{\partial z_j} = \frac{\partial C}{\partial a_j} \frac{\partial a_j}{\partial z_j}$.

We would like to show:

- $\frac{\partial a_j}{\partial z_j} = f'(z_j)$ (1)
- $\frac{\partial C}{\partial a_j} = \sum_k w_{kj} \delta_k$ (2)

For (1), we can write:

- $\frac{\partial a_j}{\partial z_j} = \frac{\partial f(z_j)}{\partial z_j} = f'(z_j)$ (3)

Now, for (2), we can write, using the chain rule and knowing that the hidden layer j is connected to output layer k:

- $\frac{\partial C}{\partial a_j} = \frac{\partial C}{\partial a_k} \frac{\partial a_k}{\partial z_k} \frac{\partial z_k}{\partial a_j}$ (4)

From that:

- $\frac{\partial C}{\partial a_k} \frac{\partial a_k}{\partial z_k} = \delta_k$ from the definition of δ_k (5)
- $\frac{\partial z_k}{\partial a_j} = \frac{\partial (w_{kj} a_j)}{\partial a_j} = w_{kj}$ (6)

And now (4) becomes:

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

In order to generalize for all the output layers, we can re-write (4) as:

- $\frac{\partial C}{\partial a_j} = \sum_k w_{kj} \delta_k$ (7)

Finally, using (3) and (7), we can write the desired equation:

- $\delta_j = f'(z_j) \sum_k w_{kj}\delta_k$

$\square$

Task 1b)

We want to re-write $w_{ji} := w_{ji} - \eta \frac{\partial C}{\partial w_{ji}}$ and $w_{kj} := w_{kj} - \eta \frac{\partial C}{\partial w_{kj}}$ in matrix notation, for the whole layer.

Assume that the input layer has I neurons, the hidden layer has J neurons and the output layer has K neurons.

We start with W^J for the hidden layer.

- $W^J := W^J - \eta \frac{\partial C}{\partial W^J}$

Where W^J is the matrix of weights connecting the input layer to the hidden layer, and $\frac{\partial C}{\partial W^J}$ is the matrix of the partial derivatives of the cost function with respect to the weights connecting the input layer to the hidden layer.

The sizes are as follows:

- W^J is of size $J \times I$
- $\frac{\partial C}{\partial W^J}$ is of size $J \times I$

Now, for the output layer W^K:

- $W^K := W^K - \eta \frac{\partial C}{\partial W^K}$

Where W^K is the matrix of weights connecting the hidden layer to the output layer, and $\frac{\partial C}{\partial W^K}$ is the matrix of the partial derivatives of the cost function with respect to the weights connecting the hidden layer to the output layer.

The sizes are as follows:

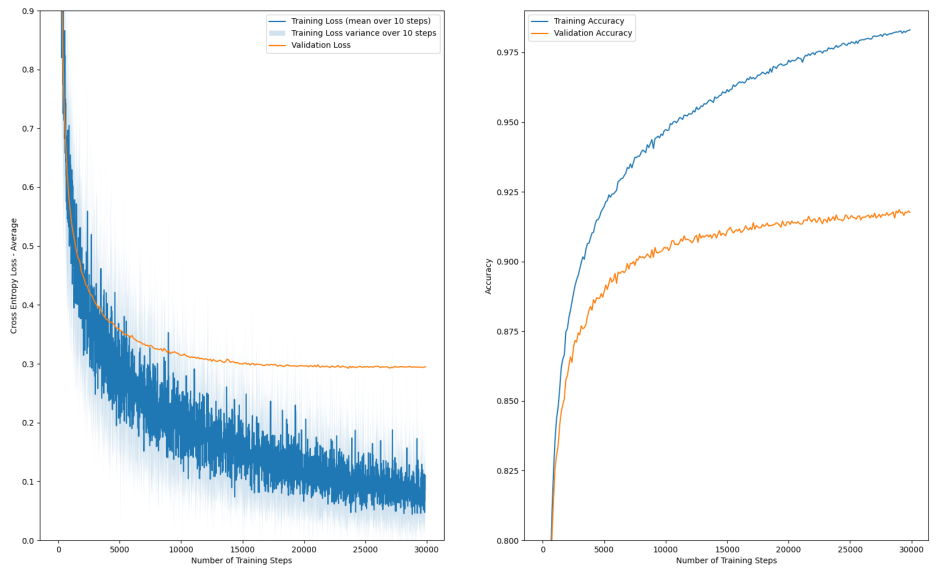
- W^K is of size $K \times J$
- $\frac{\partial C}{\partial W^K}$ is of size $K \times J$

Task 2

Task 2a)

The mean is μ: 33.55274553571429 and the standard deviation is σ: 78.87550070784701.

Task 2c)



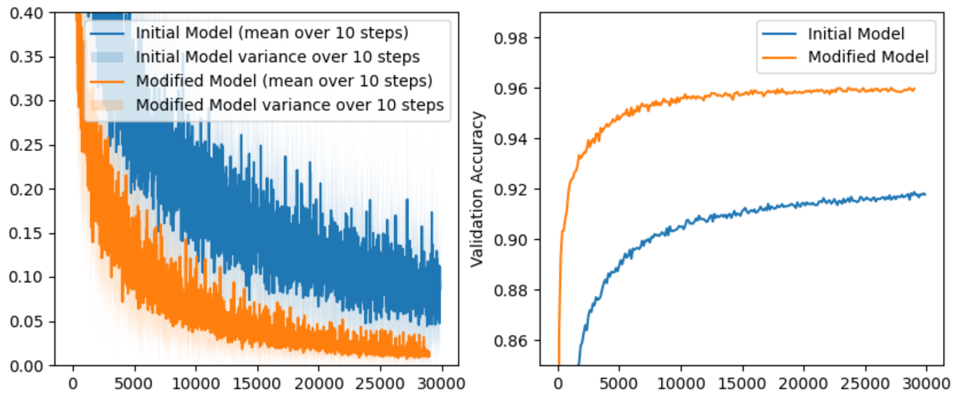
Task 2d)

Our network has 785 input nodes, 64 hidden nodes and 10 output nodes.

Our parameters in this case are $785 * 64 + 64 * 10 = 50880$.

Task 3

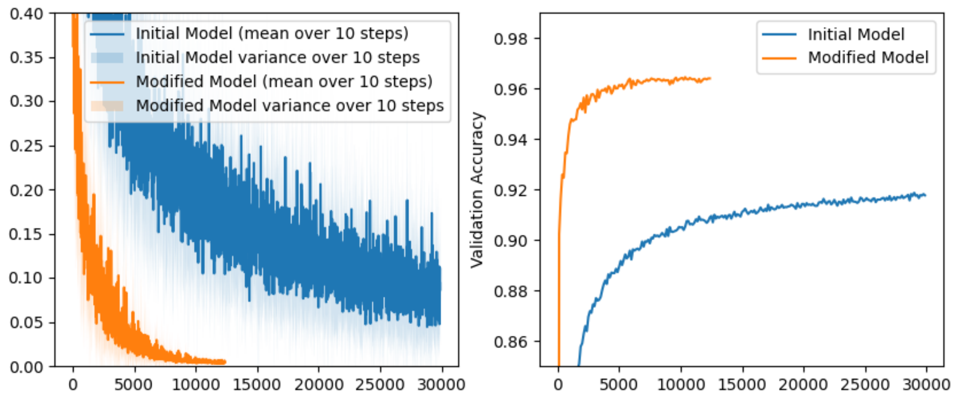
Task 3a) Weight Improvement



We can see that in the modified model the validation loss is significantly lower compare to the base model - something that is reflected in the accuracy as well.

Initializing the weights from a normal distribution with a standard deviation of 1 means our weights are not too small or too large, and thus the learning process is more efficient and the model is able to learn better as it focuses on the linear part of the sigmoid function. Additionally, having them zero-centered decreases any potential bias in the learning process.

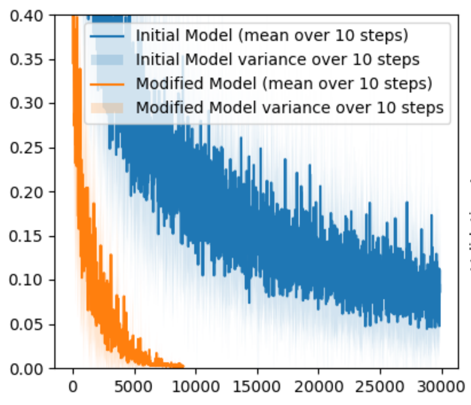
Task 3b) Improved Sigmoid w/ Weight Improvement

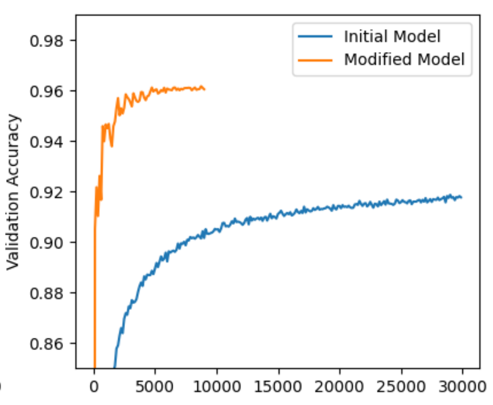


Using the improved sigmoid function in combination with the weights initialization trick, we can see that the validation loss is significantly lower compare to the base model - something that is reflected in the accuracy as well.

The improved sigmoid produces outputs that are closer to zero, thus providing the next layers with normalized inputs. The linear term 1.7159 helps us avoid flat regions in the sigmoid function, and thus the learning process is more efficient as we notice a faster convergence.

Task 3c) Momentum w/ Improved Sigmoid and Weight Improvement





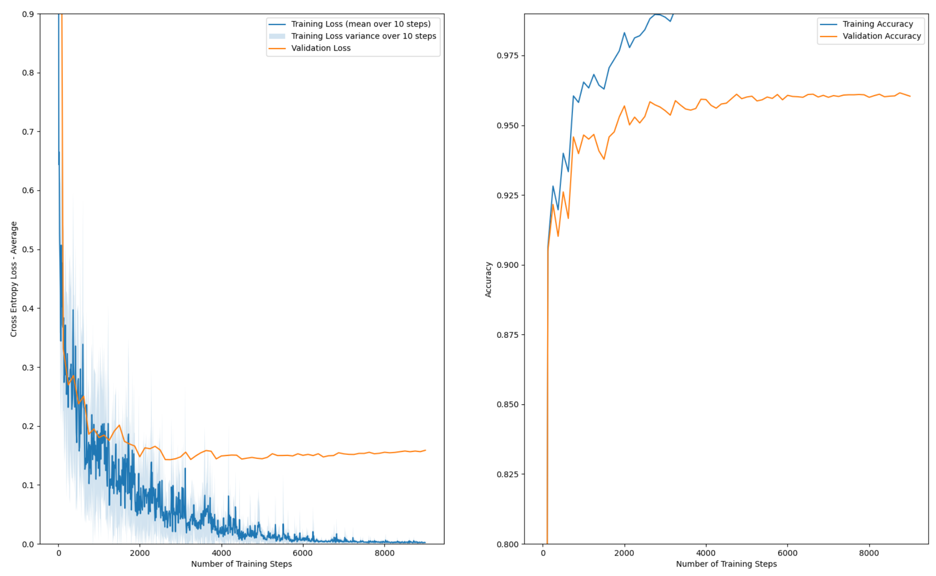
Finally, using the momentum in combination with the improved sigmoid function and the weights initialization trick, we can see that the validation loss is significantly lower compared to the base model and an even faster convergence than previously achieved.

The loss flattens out much quicker than before, and the accuracy is higher than before. This is due to the fact that the momentum allows the gradient to descent faster in flat regions and slower in steep regions, thus avoiding local minima and converging faster.

To sum, these Tricks of the Trade significantly improve both the accuracy and the convergence speed of the model - and their benefits are cumulative. However, generalization is not really improved as we notice a large gap between the training and validation loss and accuracy - thus overfitting is still an issue.

Task 4

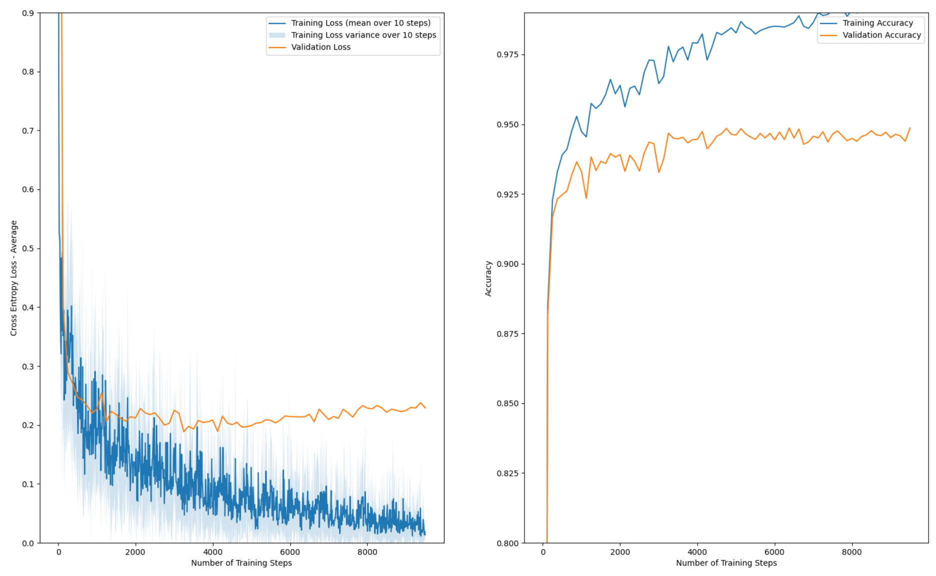
Task 4 - Default Model Reference



We can see that the default model with **1 hidden layer and 64 hidden units** has a validation loss of around 0.15 and a training loss of around 0.002. The accuracy is 0.96 and 1.0 respectively.

The improvements implemented earlier are enabled in this model.

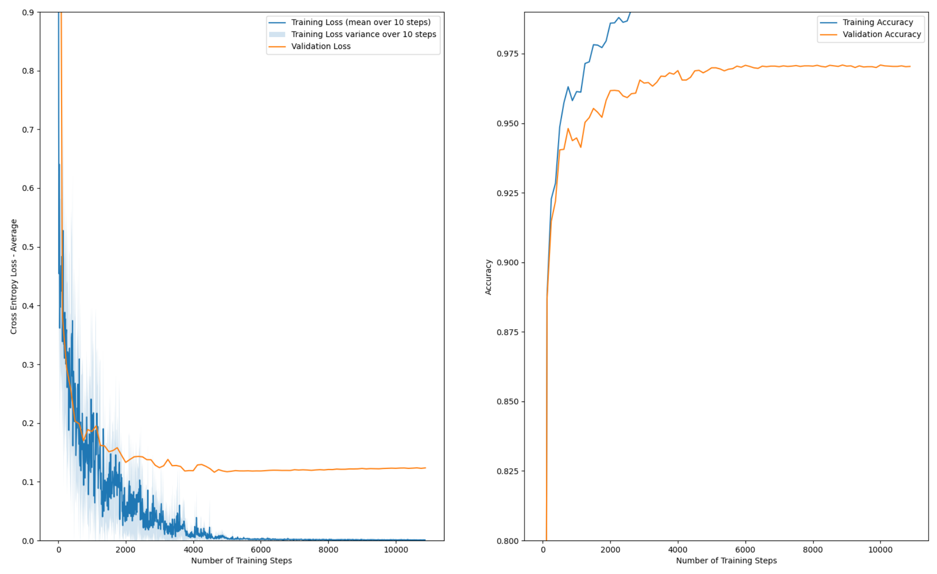
Task 4a)



For 32 hidden units, we can see that the validation loss is around 0.22 and the training loss is around 0.022. The accuracy is 0.94 and 0.99 respectively.

We see a slight decrease in the accuracy and an increase in the loss, which is expected as the model has less capacity to learn due to the reduced number of hidden units. Specifically, the model is not able to capture the complexity of the data as well as before.

Task 4b)

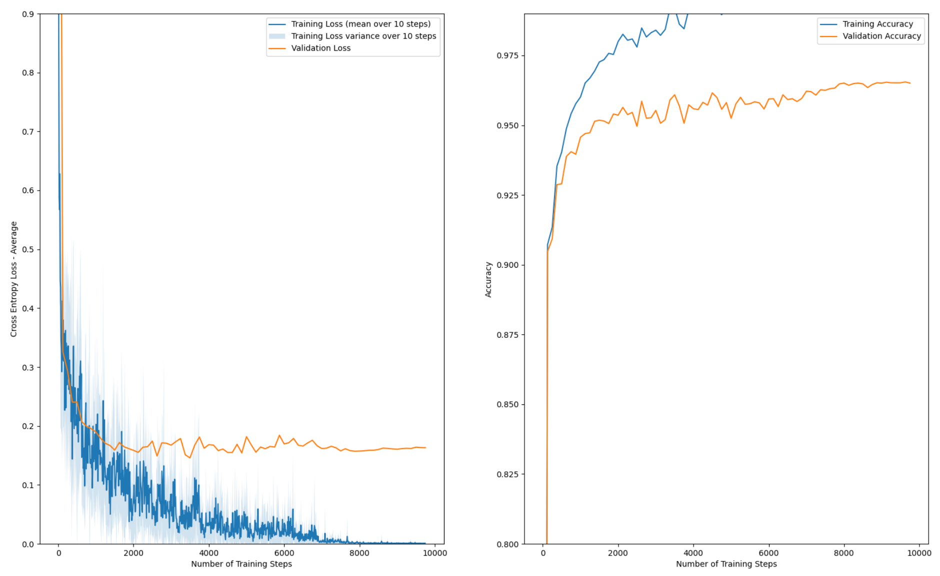


For 128 hidden units, we can see that the validation loss is around 0.12 and the training loss is around 0.0008. The accuracy is 0.97 and 1.0 respectively.

While the accuracy and the training loss are both improved, the validation loss is not *significantly* improved. This is due to the fact that the model is overfitting the training data by capturing too much of its complexity, and thus it is not able to generalize well to the validation data.

Task 4d)

In task 3, our model had 50880 parameters. In order to replicate this amount of total parameters with 2 equally sized hidden layers instead of 1, we'd need to make them of size 59 each. This is because the input layer has 785 neurons, and the output layer has 10 neurons. Thus, the total number of parameters is $785 * 59 + 59 * 59 + 59 * 10 = 50386$.

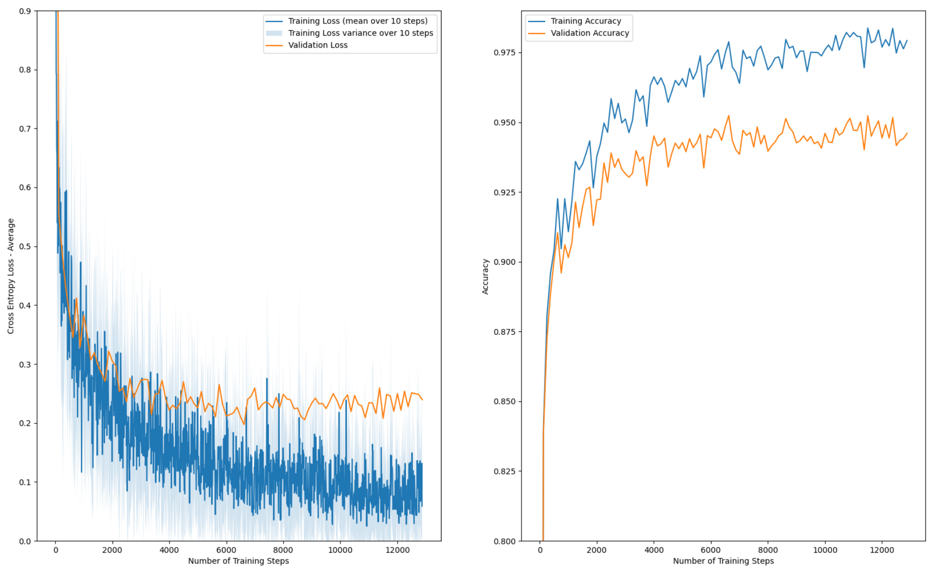


We can see that the validation loss is around 0.16 and the training loss is around 0.0008. The accuracy is 0.96 and 1.0 respectively.

The model performs slightly worse (negligible) than the 128 hidden unis - 1 hidden layer one. This is because training loss and accuracy have peaked and the model was already overfitting the training data. Validation, however, has very slightly worsened, therefore the model is not able to generalize as well as before.

Task 4e)

We trained for 25 epochs, although early stopping kicked in at 20.



We notice that the validation loss is around 0.23 and the training loss is around 0.07. The accuracy is 0.97 and 0.94 respectively.

This model, despite its impressive size, is overfitting so much that the performance has actually deteriorated instead of plateauing. Due to the amount of hidden layers, the weights take longer to converge to a minimum, and not only is training slower but the model does not generalize well at all as it deeply tries to replicate the training data at every training step.