TDT4265 - Computer Vision and Deep Learning

Assignment 1

Martin Johannes Nilsen

3rd February 2022

Task 1 - Regression theory

Task 1a

In this task, we were to derive the gradient for Logistic Regression. We have that the cross entropy loss for one data point is given by the equation

$$C^n = -\left(y^n \ln\left(\hat{y}^n\right) + \left(1 - y^n\right) \ln\left(1 - \hat{y}^n\right)\right)$$

Based on these two facts

$$\frac{\partial f\left(x^n\right)}{\partial w_i} = x_i^n f\left(x^n\right)\left(1 - f\left(x^n\right)\right),$$
$$\hat{y}^n = f\left(x^n\right)$$

We can use the chain rule for derivating the equation. The end goal is to prove that the gradient for the equation is

$$\frac{\partial C^n(w)}{\partial w_i} = -\left(y^n - y^n\right) x_i^n$$

We start off by calculating $\frac{\partial C^n(w)}{\partial f(x^n)}$ in two steps

$$\frac{\partial C^n(w)}{\partial f\left(x^n\right)} = \frac{\partial C^n(w)}{\partial \hat{y}^n}$$
$$\frac{\partial C^n(w)}{\partial \hat{y}^n} = \frac{\hat{y}^n - y^n}{\left(1 - \hat{y}^n\right)\hat{y}^n}$$

From the chain rule we now have that

$$\frac{\partial C^n(w)}{\partial w_i} = \frac{\partial C^n(w)}{\partial f\left(x^n\right)} * \frac{\partial f\left(x^n\right)}{\partial w_i}$$

Which further on gives us the following equation

$$\frac{\partial C^n(w)}{\partial w_i} = \frac{\hat{y}^n - y^n}{\left(1 - \hat{y}^n\right)\hat{y}^n} * x_i^n f\left(x^n\right)\left(1 - f\left(x^n\right)\right)$$

And, since $\hat{y} = f(x)$ we get the following equation

$$\frac{\partial C^n(w)}{\partial w_i} = \underline{\underline{-\left(y^n - \hat{y}^n\right) x_i^n}}$$

which is the same as the wanted equation.

Task 1b

In this task we are given the following vector $\hat{y}$ (with length K), where each element $\hat{y}_k$ represents the probability that x is a member of class k

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}}$$

We are to derive this and show that the gradient is

$$\frac{\partial C^n(w)}{\partial w_{kj}} = -x_j^n \left(y_k^n - \hat{y}_k^n\right)$$

As said in one of the hints, we want to break down the derivation of the softmax into two stages, namely $k = k'$ and $k \neq k'$

For the first of the stages, $k = k'$, we have that

$$\begin{aligned}
\frac{\partial \hat{y}_k}{\partial z_k} &= \frac{e^{z_k} * \sum_{k'}^{K} e^{z'_k} - e^{z_i} * e^{z_k}}{\left(\sum_{k'}^{K} e^{z'_k}\right)^2} \\
&= \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}} * \frac{\sum_{k'}^{K} e^{z'_k} - e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}} \\
&= \hat{y}_k \left(1 - \hat{y}_k\right)
\end{aligned}$$

And the other case, $k \neq k'$

$$\begin{aligned}
\frac{\partial \hat{y}_k}{\partial z_{k'}} &= \frac{0 - e^{z_k} e^{z'_k}}{\left(\sum_{k'}^{K} e^{z'_k}\right)^2} \\
&= -\hat{y}_k \hat{y}_{k'}
\end{aligned}$$

We can now use these to simplify our equation when derivating the cross entropy loss function, which for multiple classes is defined as

$$C(w) = \frac{1}{N} \sum_{n=1}^{N} C^n(w)$$

where we have that

$$C^n(w) = -\sum_{k=1}^{K} y_k^n \ln \hat{y}_k^n$$

$$\frac{\partial C^n(w)}{\partial z_k} = -\frac{\partial y_k}{\partial z_k} * \frac{y_k}{\hat{y}_k} - \sum_{k \neq k'}^{K} \frac{y_{k'}}{\hat{y}_k} * \frac{\partial y_k}{\partial z_{k'}}$$

We simplify the equation with the result from the former derivation

$$\frac{\partial C^n(w)}{\partial z_k} = -y_k + \hat{y}_k \sum_{k=1}^{K} y_k$$

$$= -y_k + \hat{y}_k$$

By utilizing the z_k we are given in the assignment as

$$z_k = \sum_{i}^{I} w_{k,i} * x_i$$

We get the following last steps

$$\frac{\partial z_k}{\partial w_{k,i}} = x_i$$

$$\frac{\partial C^n(w)}{\partial w_{k,i}} = \frac{\partial z_k}{\partial w_{k,i}} * \frac{\partial C^n(w)}{\partial z_k}$$

$$= x_i * (-y_k + \hat{y}_k)$$

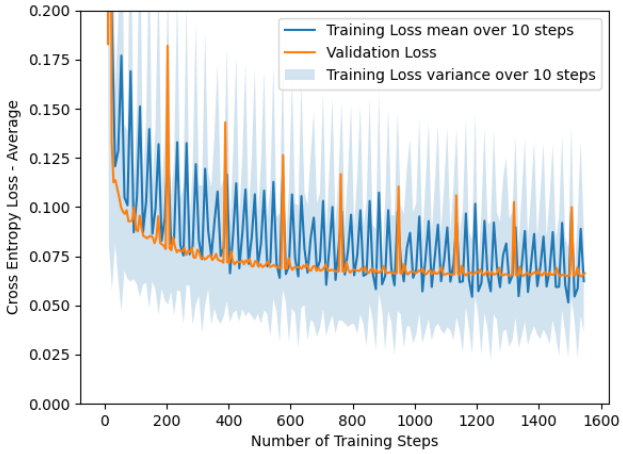
ending up with this answer

$$= \underline{\underline{x_i^n \, (y_k^n - \hat{y}_k^n)}}$$

which is the same as what we was expected to show.

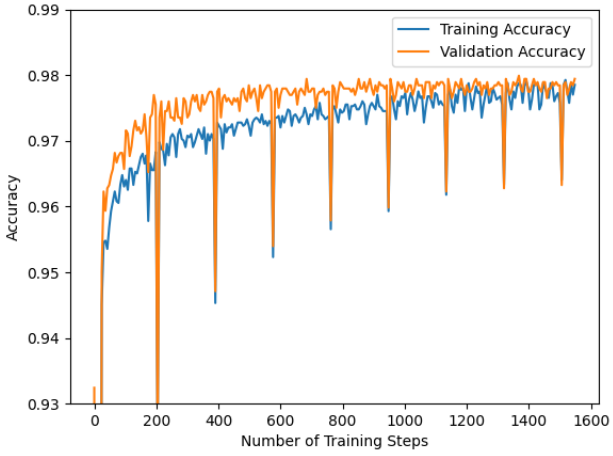
Task 2 - Logistic regression by gradient descent

Task 2b



In the plot above, we can see the plotted training and validation loss decreasing, in addition to the training loss varying less and less. In the further tasks, we want to decrease these spikes we see here, which can be fixed by shuffling the dataset. We are also going to implement early stopping.

Task 2c

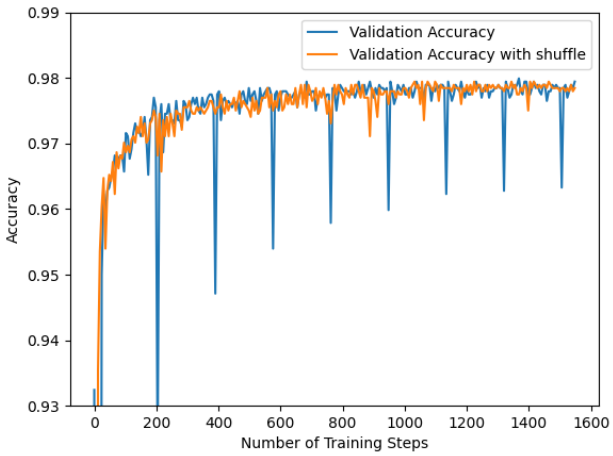


In the figure above we see the training and validation accuracy plotted in the same training as the plot from last subtask. Also here we get the spikes, which is a good indicator that the functions are implemented in the correct way. Early stopping and data shuffling are next up!

Task 2d

The early stopping kicks in at epoch 33. It should be mentioned that I had a conflict with my own thoughts regarding whether early stopping should be based off the last ten losses, or the overall ten best losses. I ended up using the minimum of the tracked loss-values in the dictionary val_history["loss"], and used the .values()-function for fetching the values as a list. This also seems to be the best alternative as this won't allow the loss to fluctuate between two values, which it could have done if we chose to only look at the previous loss and see if the new one is better or not. On the other side, it could be a problem that the model stops too early because of the natural fluctuations during training. This could be an argument of the last of the two presented alternatives.

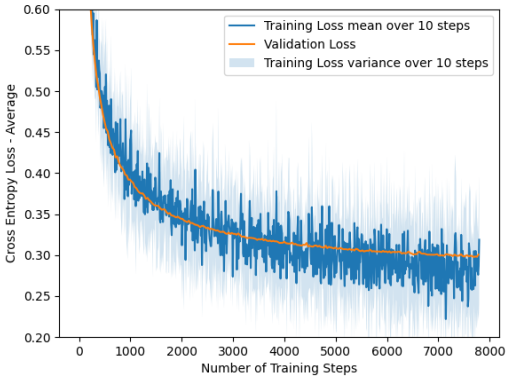
Task 2e



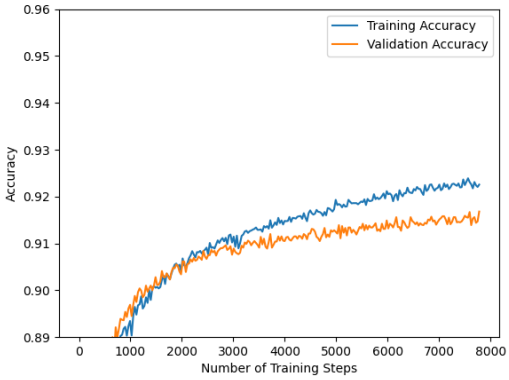
As we can see in the figure above, the implementation of shuffling reduces the spikes by a lot. In the binary case, as we have with the logistic regression with only 2 and 3 as targets, there could be a generally high successrate classifying the label which occur the most. This could lead to periodic drops in accuracy and corresponding spikes in the cost/loss function. As we may observe in the figure above, the spikes do decrease as the training proceeds, so what seems to be a naive guessing occurs less frequently as the model is further trained. With shuffling enabled, the pattern is no longer predictable, thus the periodic spikes disappear.

Task 3 - Softmax regression by gradient descent

Task 3b



Task 3c



Task 3d

From the accuracy plot presented in 3c, it seems that the training accuracy outperforms the validaton after approximately 3000 training steps. Generally, when we are observing that the training accuracy increases at a higher rate than the validation accuracy, with the validation accuracy stagnating, it may be a sign of overfitting - but as it still increases I would not call it overfitting just yet. Seeing this tendency is less clear if you observe the figure of loss, but one may argue that the training loss outperforms the validation loss in this case as well. So to summarize, I would say there are clear tendencies here. Furthermore, If training continued we could have seen the validation loss converging or even start increasing while the training loss keeps decreasing. This would have been a clear sign of overfitting.

Task 4 - Regularization

Task 4a[1]

We have been given the following equation for regularization

$$J(w) = C(w) + \lambda * R(w)$$

and the funtion of the complexity penalty R as

$$R(w) = \frac{1}{2} \sum_{i,j} w_{i,j}^2$$

We already have the gradient for the softmax loss function from 1b, given as

$$\frac{\partial C(w)}{\partial w_{k,j}} = -\left(y_k^n - \hat{y}_k^n\right) x_j^n$$

Therefore, we are only yet to figure out the regularization derivative, which makes us able to simplify the following equation

$$\frac{\partial J(w)}{\partial w} = \frac{\partial C(w)}{\partial w} + \lambda * \frac{\partial R(w)}{\partial w}$$

We solve $\frac{\partial R(w)}{\partial w_{k,j}}$

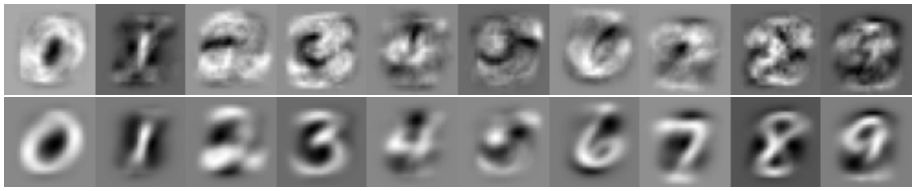
$$\frac{\partial R(w)}{\partial w_{k,j}} = \frac{\partial \left(\frac{1}{2} \sum_{k,j} w_{k,j}^2\right)}{\partial w_{k,j}} = w_{k,j}$$

and end up with the following equation

$$\frac{\partial J(w)}{\partial w_{k,j}} = \underline{\underline{-\left(y_k^n - \hat{y}_k^n\right) x_j^n + \lambda w_{k,j}}}$$

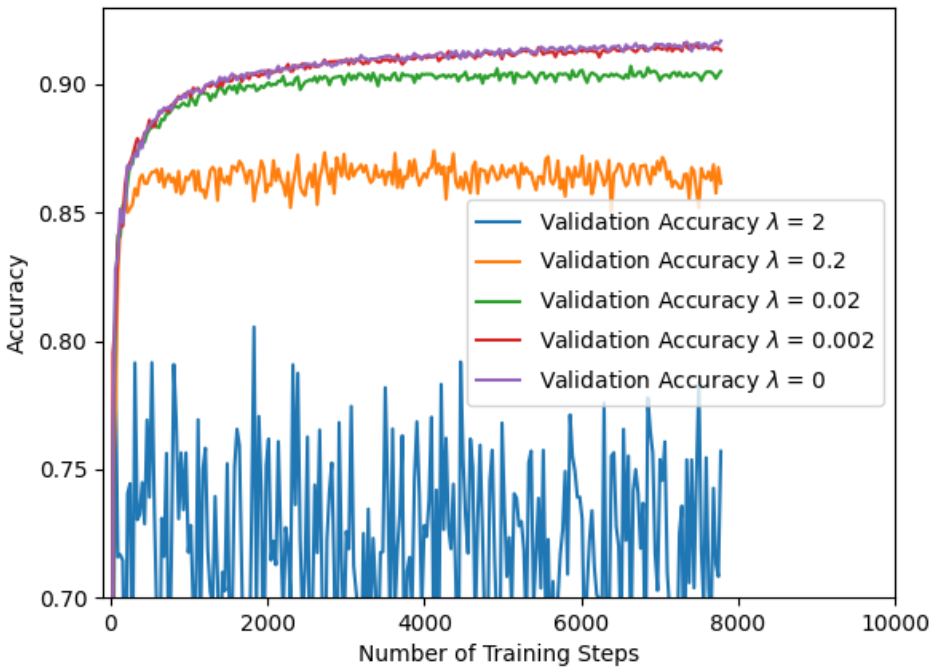
[1]In the original pull from the repository, the assignment pdf had the regularization defined as $R(w) = \sum_{i,j} w_{i,j}^2$ and not $R(w) = \frac{1}{2} \sum_{i,j} w_{i,j}^2$. My original answer was $-\left(y_k^n - \hat{y}_k^n\right) x_j^n + 2\lambda w_{k,j}$

Task 4b



In the two figures above, the weights of a trained model with regularization is plotted. The parameter λ used for the models are 0 in the top row, and 2 in the bottom row. In the top row, the regularization is therefore non-existent. One can clearly observe that the top row has more noise than the other. The point of regularization is to introduce some form of generalization to the model. In general, that would mean an increase in noise as this is a sign of a better generalized model. However, in this example we see that the weights without generalization has more noise, and thus look to be the most general. The bottom row looks more blended. In other words, the regularization punishes the model for having weights with large magnitude, i.e. complexity. One can argue that this comes from the fact that the model only have one layer, thus is a very simple model, and that if the model had been more complex with more layers, things could have been different and you would prefer a larger λ.

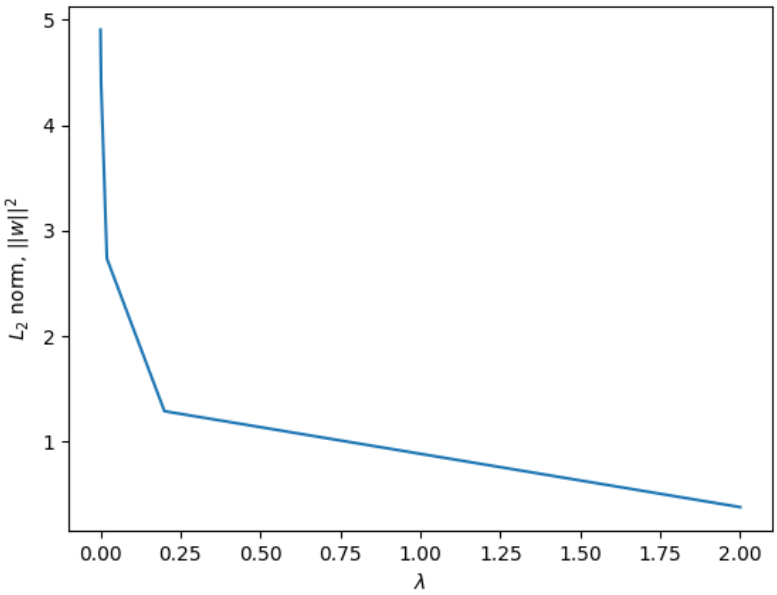
Task 4c



Task 4d

It is clear that the validation accuracy decreases when increasing λ, which is expected behaviour based on what was noticed in task 4b. One possible explanation is that the model becomes too simple with weights that are too small for the classification task, when the wanted effect of regularization, making the model simpler, has taken place. Thus the performance becomes worse when increasing the regularization.

Task 4e



The figure is illustrating the length of the weight vector (L_2 norm) for each corresponding λ-value used in task 4. In this illustration, we clearly see the effect mentioned above. The regularization simplifies the model and thus also reduces the norm of the weights. Therefore, even though the regularization does exactly as intended, it does not yield good results in our case.