TDT4265 - Computer Vision and Deep Learning
Assignment 1

Jakob Vahlin
Kristian Stensgård

Contents

1 Theory

1.1 Task 1a: Derive the gradient for Logistic Regression

We wish to derive the gradient of the **cross entropy loss** for a training set n, defined as

$$C^n(w) = -\big(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y^n})\big) \tag{1}$$

where $\hat{y}^n = f(x^n) = \frac{1}{1+e^{-w^T x^n}}$ is the Sigmoid activation function.

To simplify the calculations, we can split the cross entropy loss function, $C^n(w)$ into two parts,

$$C^n(w) = -\big(g^n(w) + h^n(w)\big) = -\big(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y^n})\big) \tag{2}$$

by defining $g^n(w) := y^n \ln(\hat{y}^n)$ and $h^n(w) := (1-y^n)\ln(1-\hat{y^n})$. The two functions can now be differentiated separately, and their results added, due to the fact that differentiation is a linear operator. When differentiating the two functions, $g^n(w)$ and $h^n(w)$, the derivative of $f(x^n)$ w.r.t the variable w_i, given by (3) will be used

$$\frac{\partial f(x^n)}{\partial w_i} = x_i^n f(x^n)(1-f(x^n)) \tag{3}$$

Differentiating $g^n(w) = y^n \ln \hat{y}^n = y^n \ln f(x^n)$ gives:

$$\begin{aligned}
\frac{\partial}{\partial w_i} g^n(w) &= \frac{\partial}{\partial w_i} y^n \ln f(x^n) && && (4)\\
&= y^n\Big(\frac{1}{f(x^n)} x_i^n f(x^n)(1-f(x^n))\Big) && \text{By the chain rule and (3)} && (5)\\
&= y^n x_i^n (1-f(x^n)) && \text{Cancelling terms} && (6)
\end{aligned}$$

Differentiating $h^n(w) = (1-y^n)\ln(1-\hat{y^n}) = (1-y^n)\ln(1-f(x^n))$ gives:

$$\begin{aligned}
\frac{\partial}{\partial w_i} h^n(w) &= \frac{\partial}{\partial w_i}(1-y^n)\ln(1-f(x^n)) && && (7)\\
&= -(1-y^n)\frac{1}{1-f(x^n)} x_i^n f(x^n)(1-f(x^n)) && \text{By the chain rule and (3)} && (8)\\
&= -(1-y^n) x_i^n f(x^n) && \text{Cancelling terms} && (9)
\end{aligned}$$

Finally, combining the results from (6) and (9) we obtain the gradient of the cross entropy loss:

$$\begin{aligned}
\frac{\partial}{\partial w_i} C^n(w) &= -\Big(\frac{\partial}{\partial w_i} g^n(w) + \frac{\partial}{\partial w_i} h^n(w)\Big) && (10)\\
&= -x_i^n(y^n - \hat{y^n}) && (11)
\end{aligned}$$

1.2 Task 1b: Derive the gradient for Softmax Regression

We wish to derive the gradient of the multiple class cross entropy loss function, $C^n(w)$, defined as

$$C^n(w) = -\sum_{k=1}^{K} y_k^n \ln \hat{y_k^n} \tag{12}$$

where $\hat{y_k^n} = \frac{e^{z_k n}}{\sum_{k'}^{K} e^{z_{k'}^n}}$, with $z_k^n = w_k^T \cdot x^n = \sum_i w_{k,i} \cdot x_i^n$. To simplify the differentiation $C^n(w)$ can be simplified in the following way

$$
\begin{aligned}
C^n(w) &= -\sum_{k=1}^{K} y_k^n \ln \hat{y_k^n} && (13)\\
&= -\sum_{k=1}^{K} y_k^n \ln \left(\frac{e^{z_k^n}}{\sum_{k'}^{K} e^{z_{k'}^n}} \right) && (14)\\
&= -\sum_{k=1}^{K} y_k^n \left(\ln (e^{z_k^n}) - \ln \left(\sum_{k'}^{K} e^{z_{k'}^n} \right) \right) && \text{Logarithm quotient rule} \quad (15)\\
&= -\sum_{k=1}^{K} y_k^n z_k^n + \sum_{k=1}^{K} y_k^n \ln \left(\sum_{k'}^{K} e^{z_{k'}^n} \right) && \ln(e^a) = a \quad (16)\\
&= -\sum_{k=1}^{K} y_k^n z_k^n + \ln \left(\sum_{k'}^{K} e^{z_{k'}^n} \right) && \text{Sum of } y^n \text{ entries equals 1} \quad (17)
\end{aligned}
$$

Similarly to Task 1a the expression in (17) can be expressed as the sum of two functions. By defining $g^n(w) := \sum_{k=1}^{K} y_k^n z_k^n$ and $h^n(w) := \ln \left(\sum_{k'}^{K} e^{z_{k'}^n} \right)$ (17) can be written as

$$
\begin{aligned}
C^n(w) &= -\sum_{k=1}^{K} y_k^n z_k^n + \ln \left(\sum_{k'}^{K} e^{z_{k'}^n} \right) && (18)\\
&= -g^n(w) + h^n(w) && (19)
\end{aligned}
$$

Differentiating $g(w)$ gives

$$
\begin{aligned}
\frac{\partial}{\partial w_{kj}} g(w) &= \frac{\partial}{\partial w_{kj}} \sum_{k=1}^{K} y_k z_k && (20)\\
&= \frac{\partial}{\partial w_{kj}} \sum_{k=1}^{K} y_k \sum_{i=1}^{785} w_{ki} x_i && \text{By definition of } z_k \quad (21)
\end{aligned}
$$

By writing out a few terms of the sums in (21) we obtain

$$
\begin{aligned}
\frac{\partial}{\partial w_{kj}} g(w) = \frac{\partial}{\partial w_{kj}} \Big(& y_1(w_{11}x_1 + w_{12}x_2 + \cdots + w_{1i}x_i) \\
& + y_2(w_{21}x_1 + w_{22}x_2 + \cdots + w_{2i}x_i) \\
& + y_k(w_{k1}x_1 + w_{k2}x_2 + \cdots + w_{ki}x_i) \Big)
\end{aligned} \qquad (22)
$$

The derivatie w.r.t w_{jk} of any given term in (22) is only non-zero when $j = i$, in which the resutling non-zero derivative is $y_k x_j$. Thus we can conclude that

$$
\frac{\partial}{\partial w_{kj}} g(w) = y_k \cdot x_j \qquad (23)
$$

Differentiating $h(w)$ gives

$$\frac{\partial}{\partial w_{kj}} h(w) = \frac{\partial}{\partial w_{kj}} \ln\left(\sum_{k'}^{K} e^{z_{k'}}\right) \tag{24}$$

$$= \frac{\partial}{\partial w_{kj}} \ln\left(\sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i}\right) \qquad \text{By definition of } z'_k \tag{25}$$

$$= \frac{1}{\sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i}} \frac{\partial}{\partial w_{kj}} \sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i} \qquad \text{By the chain rule} \tag{26}$$

By writing out the sums of the remaining expression to be differentiated in (26) we obtain

$$\frac{\partial}{\partial w_{kj}} \sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i} = \frac{\partial}{\partial w_{kj}} \left(e^{\sum_i w_{1i} x_i} + e^{\sum_i w_{2i} x_i} + \cdots + e^{\sum_i w_{Ki} x_i}\right) \tag{27}$$

$$\begin{aligned} &= \Big(\frac{\partial}{\partial w_{kj}} \sum_i w_{1i} x_i\Big) e^{\sum_i w_{1i} x_i} \\ &+ \Big(\frac{\partial}{\partial w_{kj}} \sum_i w_{2i} x_i\Big) e^{\sum_i w_{2i} x_i} \\ &+ \cdots + \Big(\frac{\partial}{\partial w_{kj}} \sum_i w_{Ki} x_i\Big) e^{\sum_i w_{Ki} x_i} \end{aligned} \tag{28}$$

The derivative w.r.t w_{jk} of any given term in (28) is only non-zero when $j = i$ and $k = k'$, in which the resutling non-zero derivative is $x_j e^{\sum_i w_{ki} x_i}$. Thus we get

$$\frac{\partial}{\partial w_{kj}} \sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i} = x_j e^{\sum_i w_{ki} x_i} \tag{29}$$

Inserting (29) into (26) we obtain

$$\frac{\partial}{\partial w_{kj}} h(w) = \frac{1}{\sum_{k'}^{K} e^{\sum_i w_{k'i} \cdot x_i}} x_j e^{\sum_i w_{ki} x_i} \tag{30}$$

$$= \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}} x_j \qquad \text{By definition of } z_k \tag{31}$$

$$= \hat{y_k} x_j \qquad \text{By definition of } \hat{y_k} \tag{32}$$

Finally, combining the results in (23) and (32) we obtain the gradient of the multiple class cross entrpy loss function $C^n(w)$

$$\frac{\partial}{\partial w_{kj}} C^n(w) = -\frac{\partial}{\partial w_{kj}} g^n(w) + \frac{\partial}{\partial w_{kj}} h^n(w) \tag{33}$$

$$= -y_k^n x_j^n + \hat{y_k^n} x_j^n \tag{34}$$

$$= -x_j^n (y_k^n - \hat{y_k^n}) \tag{35}$$

2 Logistic Regression through Gradient Descent

2.1 Task 2b: Training loss

Logistic regression with mini-batch gradient descent for a single layer neural network has been implemented. During training the training loss for each gradient step has been tracked. Additionally the validation loss over the whole validation set has been tracked for every time the network has progressed through 20% of the training set. The loss results are shown in Figure 1. The results show that the model converges quite fast, and the validation set showing a more consistent performance. However both sets have clearly unwanted high spikes. With the training set having a higher frequency of spikes. More details on these spikes will follow in task 2e.

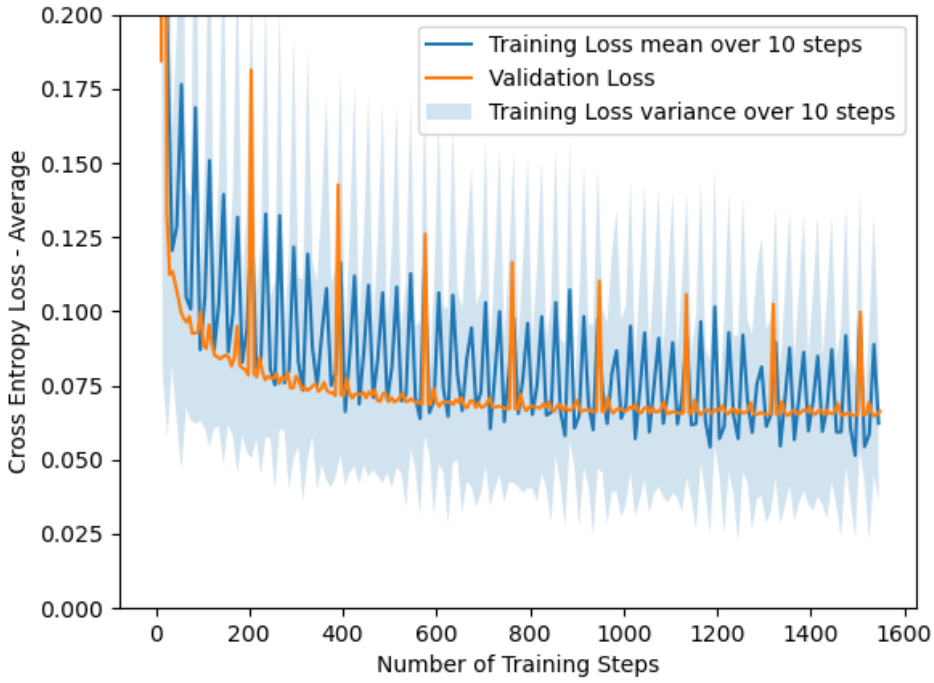

Figure 1: Mean training- and validation loss for logistic regression with mini-batch gradient descent for a single layer neural network .

2.2 Task 2c: Accuracy

A function computing the accuracy for both the training set and validation set has been implemented. The resulting accuracy plots for training and validation are shown in Figure 2. As Figure 2 shows, the accuracy for the validation and training sets converges to 98%. Note that the validation accuracy actually converges faster, than the training accuracy.

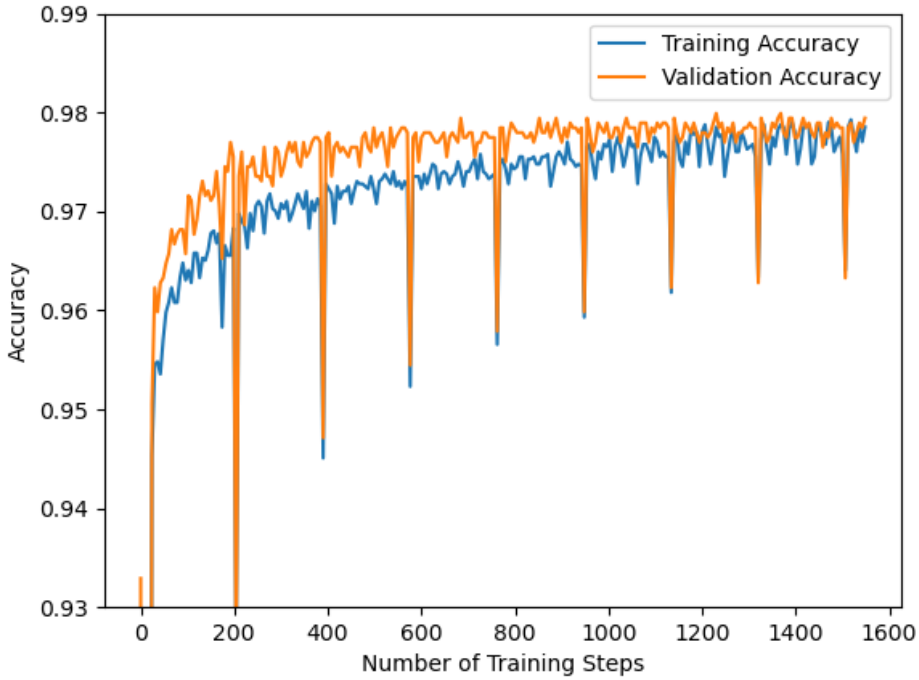

Figure 2: Accuracy for training and validation set for a given training step.

2.3 Task 2d: Early stopping

To avoid overfitting the model to the training set, *Early Stopping* has been implemented. By regularly checking if the loss for the validation set actually decreases, the training can be stopped early if the validation loss has not decreased after a given set of checks. In this implementation, the training will stop if the loss does not decrease after passing 20% of training data 10 times.

When training the model, it stopped after 33 epochs of a total 50 epochs due to the implementation of early stopping. Thus increasing the number of epochs from 50 to 500 has no practical effect.

The performance metrics for running the model in task2 with early stopping is shown in Table 1.

Final Train Cross Entropy Loss	0.071
Final Validation Cross Entropy Loss	0.066
Train accuracy	0.976
Validation accuracy	0.979

Table 1: Performance metrics for the model after stopping at 33 epochs.

2.4 Task 2e: Dataset shuffling

The network convergence can be improved by shuffling the dataset between each epoch. This way, the network does not encounter the exact same batch of examples each epoch. The effect of not shuffling the data in a neural network would be that the network memorises the training data, instead of creating a model that describes the data. This will result in overfitting and poor results on validation and test sets. By shuffling the data, different unique batches will be trained on the model, which will improve the gradient descent by having a more varied dataset.

To clearly document the effect of shuffling the dataset, the validation accuracies with and

without shuffle are shown in Figure 3. Additionally the training loss means and variances with and without shuffle are shown in Figure 4.

The shuffled training set yields less spikes in the accuracy and loss plots due to the fact that it has random orders on the training set. The spikes are due to batches that are disproportionately hard to classify. These batches, or maybe this one batch, has a set of images that are hard to classify and thus gives poor performance. By shuffling the data, we are not so unlucky to get this bad batch and therefore no huge spikes.

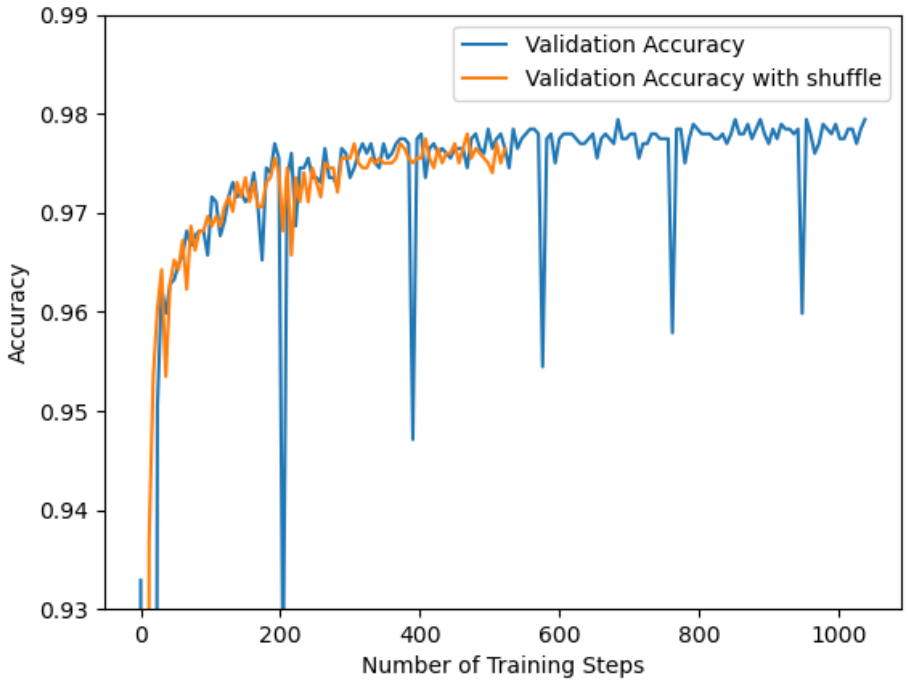

Figure 3: Accuracy of training with and without shuffled datasets.

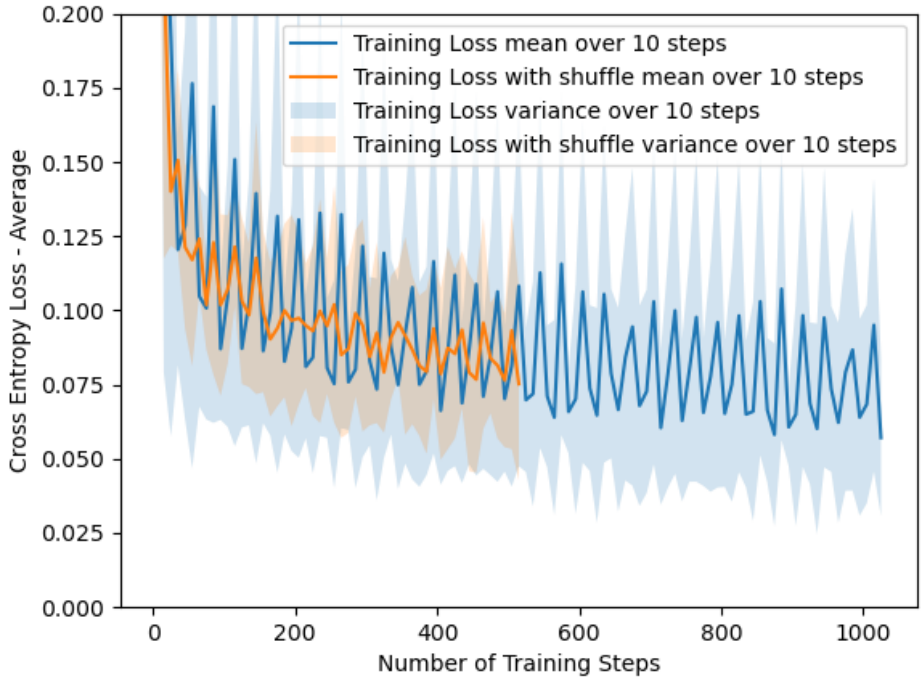

Figure 4: Training loss means and variances with and without shuffled datasets.

3 Softmax Regression through Gradient Descent

3.1 Task 3b: Training loss

Softmax regression with mini-batch gradient descent for a single layer neural network has been implemented. During training, the training loss for each gradient step has been tracked. Additionally the validation loss over the whole validation set has been tracked for every time the network has progressed through 20% of the training set. The loss results are shown in Figure 5. Here we clearly see a nice decrease in loss for the validation set with each iteration. For the training set however, even the mean over ten steps has quite high variance in loss.

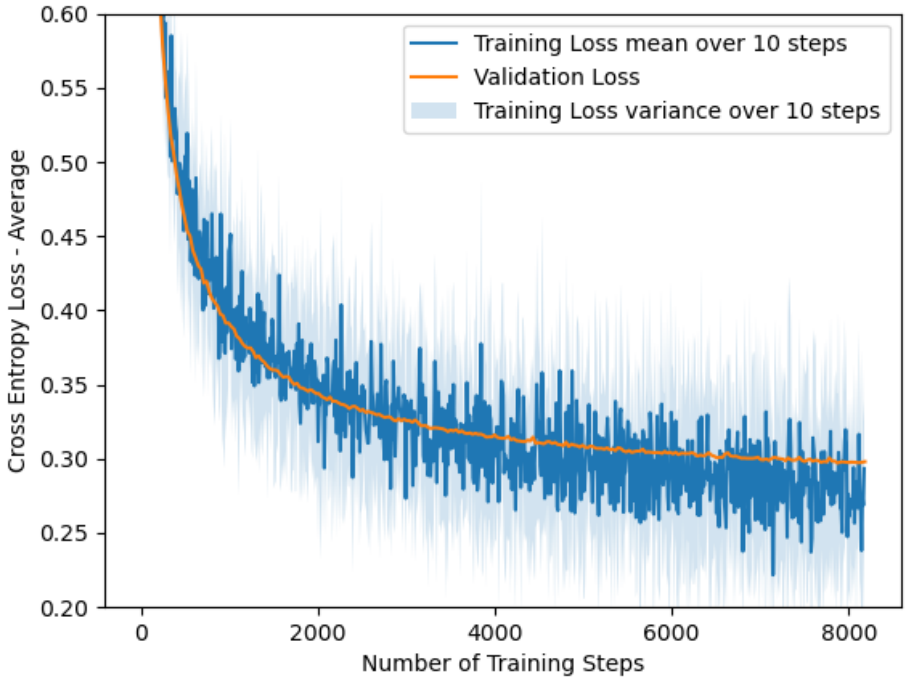

Figure 5: Training loss and validation loss for Softmax regression with mini-batch gradient descent for a single layer neural network.

3.2 Task 3c: Accuracy

A function computing the accuracy for both the training set and validation set has been implemented. The resulting accuracy plots for training and validation are shown in Figure 6. The plots show that after around 4000 training steps, the training accuracy improves significantly compared to the validation accuracy. This is further discussed in the next task.

Figure 6: Prediction accuracy for the training set and the validation set.

3.3 Task 3d: Overfitting

From the accuracy plots shown in Figure 6, it is clear that the training accuracy is significantly better then the validation accuracy, after around 4000 training steps. This could be a sign of overfitting. If the training would continue for a longer period, the model could possibly begin to show worse performance on the validation set when increasing the training, and this would be considered overfitting. However, the accuracy for the validation and training set could both converg to a single common value or two different values. This is hard to tell from the plot.

4 Regularization

4.1 Task 4a: Softmax gradient with L_2 regularization

We wish to derive the new update term for our Softmax Regression after having introduced L_2 regression, given by

$$J(w) = C(w) + \lambda R(w) \ , \ \lambda \in \mathbb{R} \tag{36}$$

where $C(w)$ is defined as

$$C(w) = \frac{1}{N} \sum_{n=1}^{N} C^n(w) \tag{37}$$

with $C^n(w)$ being the multiple class cross entropy loss function defined in (12). $R(w)$ is defined as the square of the L_2 norm of the weight matrix, w

$$R(w) = \|w\|^2 = \sum_{i,j} w_{ij}^2 \tag{38}$$

The new update term will is given by $\frac{\partial J}{\partial w}$ In other words, we wish to compute

$$\frac{\partial J}{\partial w} = \frac{\partial C(w)}{\partial w} + \lambda \frac{\partial R(w)}{\partial w} \tag{39}$$

Differentiating $C(w)$ gives:

$$\frac{\partial C(w)}{\partial w} = \frac{\partial}{\partial w} \frac{1}{N} \sum_{n=1}^{N} C^n(w) \tag{40}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w} C^n(w) \tag{41}$$

$$\tag{42}$$

Using the result from task 1b, we obtain

$$\frac{\partial C(w)}{\partial w} = \frac{1}{N} \sum_{n=1}^{N} \begin{bmatrix} \frac{\partial}{\partial w_{1,1}} C^n(w) & \frac{\partial}{\partial w_{1,2}} C^n(w) & \dots & \frac{\partial}{\partial w_{1,I}} C^n(w) \\ \frac{\partial}{\partial w_{2,1}} C^n(w) & \frac{\partial}{\partial w_{2,2}} C^n(w) & \dots & \frac{\partial}{\partial w_{2,I}} C^n(w) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial w_{K,1}} C^n(w) & \frac{\partial}{\partial w_{K,2}} \sum_{i,j} C^n(w) & \dots & \frac{\partial}{\partial w_{K,I}} C^n(w) \end{bmatrix} \tag{43}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \begin{bmatrix} -x_1^n(y_1^n - \hat{y_1^n}) & -x_2^n(y_1^n - \hat{y_1^n}) & \dots & -x_I^n(y_1^n - \hat{y_1^n}) \\ -x_1^n(y_2^n - \hat{y_2^n}) & -x_2^n(y_2^n - \hat{y_2^n}) & \dots & -x_I^n(y_2^n - \hat{y_2^n}) \\ \vdots & \vdots & \ddots & \vdots \\ -x_1^n(y_K^n - \hat{y_K^n}) & -x_2^n(y_K^n - \hat{y_K^n}) & \dots & -x_I^n(y_K^n - \hat{y_K^n}) \end{bmatrix} \tag{44}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w} C^n(w) \tag{45}$$

Differentiating $R(w)$ gives

$$\frac{\partial R(w)}{\partial w} = \frac{\partial \|w\|^2}{\partial w} \tag{46}$$

$$= \frac{\partial}{\partial w} \sum_{i,j} w_{i,j}^2 \tag{47}$$

$$= \begin{bmatrix} \frac{\partial}{\partial w_{1,1}} \sum_{i,j} w_{i,j}^2 & \frac{\partial}{\partial w_{1,2}} \sum_{i,j} w_{i,j}^2 & \dots & \frac{\partial}{\partial w_{1,I}} \sum_{i,j} w_{i,j}^2 \\ \frac{\partial}{\partial w_{2,1}} \sum_{i,j} w_{i,j}^2 & \frac{\partial}{\partial w_{2,2}} \sum_{i,j} w_{i,j}^2 & \dots & \frac{\partial}{\partial w_{2,I}} \sum_{i,j} w_{i,j}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial w_{K,1}} \sum_{i,j} w_{i,j}^2 & \frac{\partial}{\partial w_{K,2}} \sum_{i,j} w_{i,j}^2 & \dots & \frac{\partial}{\partial w_{K,I}} \sum_{i,j} w_{i,j}^2 \end{bmatrix} \tag{48}$$

Considering the matrix in (48), similarly to in task 1a and 1b, the derivative for a given matrix entry, $\frac{\partial}{\partial w_{i',j'}} \sum_{i,j} w_{i,j}^2$ is only non-zero when $i' = i$ and $j' = j$. In that case the resulting matrix entry becomes $2w_{i',j'}$. Thus we can conclude that the derivative of $R(w)$ is

$$\frac{\partial R}{\partial w} = \begin{bmatrix} 2w_{1,1} & 2w_{1,2} & \dots & 2w_{1,I} \\ 2w_{2,1} & 2w_{2,2} & \dots & 2w_{2,I} \\ \vdots & \vdots & \ddots & \vdots \\ 2w_{K,1} & 2w_{K,2} & \dots & 2w_{K,I} \end{bmatrix} \tag{49}$$

$$= 2w \tag{50}$$

where w is the weight matrix.

Finally combining (45) and (50) we obtain the new update term for the Softmax regression with L_2 regularization

$$\frac{\partial J}{\partial w} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w} C^n(w) + 2\lambda w \tag{51}$$

4.2 Task 4b: Weights for different strengths of L_2 regularization

In Figure 8 and Figure 7 we have plotted the weights of a trained model with regularization. The parameter λ used for the models is 0 and 1 respectively. Note that with $\lambda = 0$, the regularization is non existent. From the figures we clearly see that without regularization, $\lambda = 0$, the weights are more noisy. The weights with regularization, $\lambda = 1$ clearly resembles numbers. The point of regularization is to introduce some form of generalisation to the model, for instance noise, the goal of this is to make the model more general and not overfit. However in this example we see that the weights without regularization have the most noise, and thus look to be the most general. The regularization punishes the model for having weights with large magnitude. This is because having weights with smaller magnitude results in a simpler model, the model is then also considered to be more general. In our case we have a very simple network with only one layer. Therefore the model is simply to simple and the weights to sparse, when λ is large. However, if the model was more complex, more layers, things might have been different, and a larger λ would be beneficial.

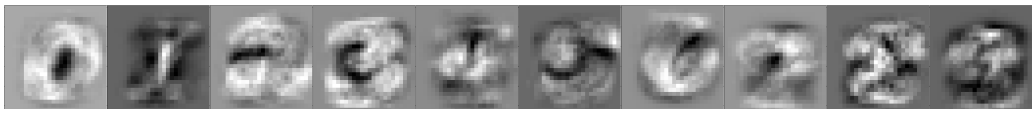

Figure 7: Combined 28x28 plot of all the weights without biases with $\lambda = 0$

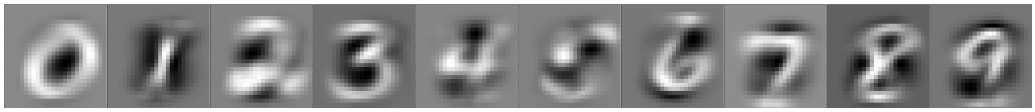

Figure 8: Combined 28x28 plot of all the weights without biases with $\lambda = 1$

4.3 Task 4c: Validation accuracy for different strengths of L_2 regularization

The accuracy for the model with the given values of lambda $\lambda = \{0, 0.001, 0.01, 0.1, 1\}$ is shown in Figure 9.

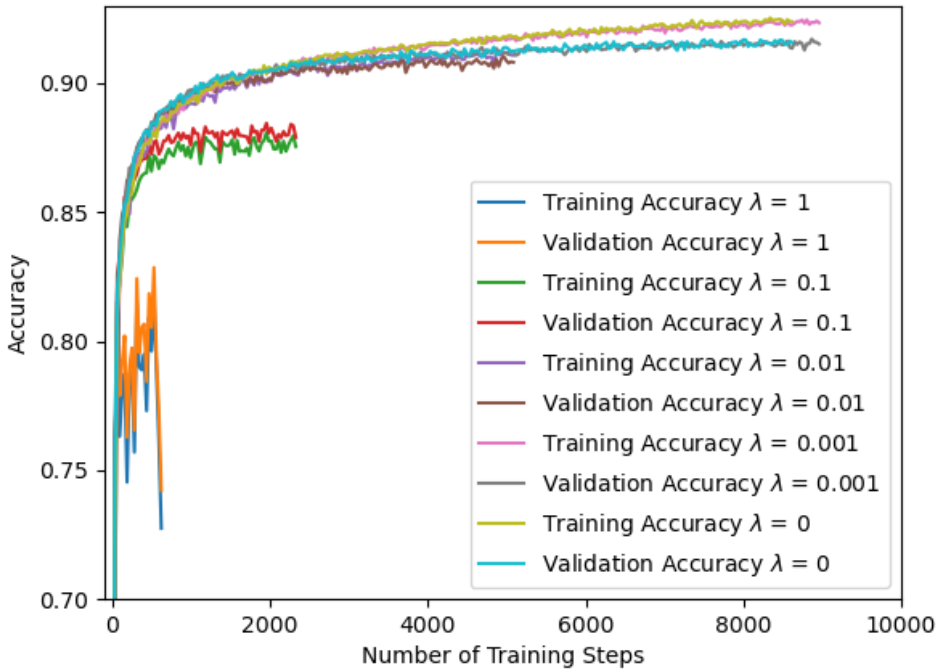

Figure 9: Accuracies for a trained model with regularisation for different values of λ

4.4 Task 4d: Degrading validation accuracy for different strengths of L_2 regularization

It is clear that the accuracy for both sets, training and validation, degrades when increasing λ. This is expected when taking in consideration what was stated in task 4b. The regularization has the wanted effect of making the model simpler, however the model becomes too simple with weights that are too small for the classification task. Thus the performance becomes worse when increasing the regularization.

4.5 Task 4e: L_2 Norm

In Figure 10 the norm of the weights is plotted against λ, here we clearly see the effect first mentioned in 4b. The regularization simplifies the model and thus also reduces the norm of w. The regularization does exactly what it is intended to do, but in our particular case, does not yield good results.

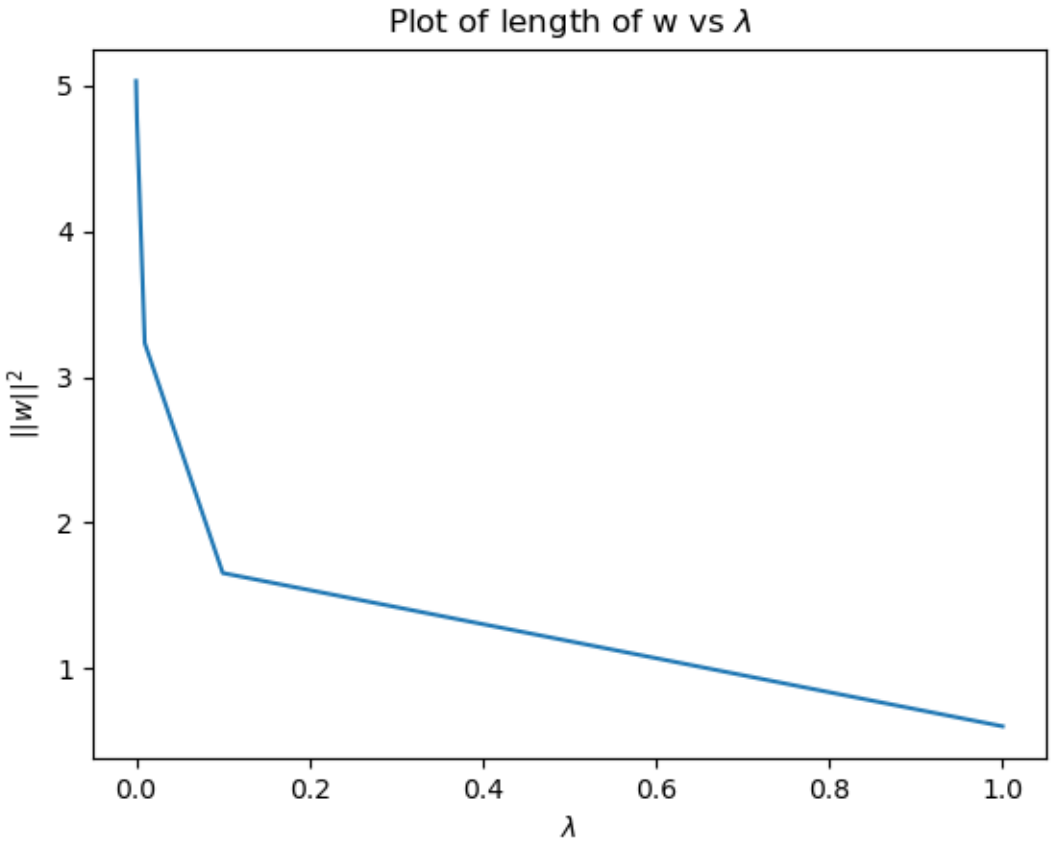

Figure 10: Plot showing the L_2 norm of the weights against the lambda values $\lambda = \{0, 0.001, 0.01, 0.1, 1\}$