Assignment 1 Report Vemund Rogne and Kristian Brudeli

This is an outline for your report to ease the amount of work required to create your report. Jupyter notebook supports markdown, and I recommend you to check out this cheat sheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet). If you are not familiar with markdown.

Before delivery, **remember to convert this file to PDF**. You can do it in two ways:

1. Print the webpage (ctrl+P or cmd+P)
2. Export with latex. This is somewhat more difficult, but you'll get somehwat of a "prettier" PDF. Go to File -> Download as -> PDF via LaTeX. You might have to install nbconvert and pandoc through conda; `conda install nbconvert pandoc`.

Task 1

task 1a)

We can get the desired gradient $\frac{\partial C^n(w)}{\partial w_i}$ by use of the chain rule:

$$\begin{aligned}
\frac{\partial C^n(w)}{\partial w_i} &= \frac{\partial C^n(w)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f} \cdot \frac{\partial f}{\partial w_i} \\
\frac{\partial C^n(w)}{\partial \hat{y}} &= -(y^n \cdot \frac{1}{\hat{y}^n} + (1 - y^n) \cdot (-1) \cdot \frac{1}{1 - \hat{y}^n}) \\
&= -\left(\frac{y^n}{\hat{y}^n} - \frac{1 - y^n}{1 - \hat{y}^n}\right)
\end{aligned}$$

$$\frac{\partial \hat{y}}{\partial f} = 1$$

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

$$\begin{aligned}
\frac{\partial C^n(w)}{\partial w_i} &= -x_i^n \hat{y}^n(1 - \hat{y}^n)\left(\frac{y^n}{\hat{y}^n} + \frac{1 - y^n}{1 - \hat{y}^n}\right) \\
&= -x_i^n(y^n(1 - \hat{y}^n) - (1 - y^n)\hat{y}^n) \\
&= -x_i^n(y^n - y^n\hat{y}^n - \hat{y}^n + y^n\hat{y}^n) \\
&= -(y^n - \hat{y}^n)x_i^n
\end{aligned}$$

task 1b)

The superscript "n" (for sample number) is implicit for all "y" and "x" variables to avoid to cluttered notation, e.g. $\hat{y} \equiv \hat{y}^n$ etc.

First, we calculate the softmax derivative w.r.t. the network outputs:

$$\frac{\partial \hat{y}_i}{\partial z_k} = \frac{\partial\hat{}}{\partial z_k}(\frac{e^{z_i}}{\sum_{k'}^K e^{z_{k'}}})$$

$$\begin{aligned}
i = k: \quad \frac{\partial \hat{y}_k}{\partial z_k} &= \frac{(\frac{\partial\hat{}}{\partial z_k} e^{z_{k'}}) \sum_{k'}^K e^{z_{k'}} - (\frac{\partial\hat{}}{\partial z_k} \sum_{k'}^K e^{z_{k'}}) e^{z_k}}{(\sum_{k'}^K e^{z_{k'}})^2} \\
&= \frac{e^{z_k}(\sum_{k'}^K e^{z_{k'}}) - e^{z_k} e^{z_k}}{(\sum_{k'}^K e^{z_{k'}})^2} \\
&= \frac{e^{z_k}}{\sum_{k'}^K e^{z_{k'}}} \cdot \frac{\sum_{k'}^K e^{z_{k'}} - e^{z_k}}{\sum_{k'}^K e^{z_{k'}}} \\
&= \hat{y}_k(1 - \hat{y}_k) \\
i \neq k: \quad \frac{\partial \hat{y}_i}{\partial z_k} &= -e^{z_i} \frac{\partial}{\partial z_k}(\frac{1}{\sum_{k'}^K e^{z_{k'}}}) \\
&= -e^{z_i} \frac{e^{z_k}}{(\sum_{k'}^K e^{z_{k'}})^2} \\
&= -\hat{y}_i^n \hat{y}_k^n
\end{aligned}$$

Using the chain rule, we have that:

$$\begin{aligned}
\frac{\partial C(w)}{w_{kj}} &= \sum_i \frac{\partial C}{\partial z_i} \frac{\partial z_i}{\partial w_{kj}} = \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial w_{kj}} + \sum_{i \neq k} \frac{\partial C}{\partial z_i} \frac{\partial z_i}{\partial w_{kj}} \\
&= \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial w_{kj}}
\end{aligned}$$

Where the last equality comes from the fact that

$$\frac{\partial z_i}{\partial w_{kj}} = 0 \quad \text{for} \quad i \neq k$$

Therefore

$$
\begin{aligned}
\frac{\partial C(w)}{\partial z_k} &= \frac{\partial}{\partial z_k}(-\sum_i^K y_i ln(\hat{y}_i)) \\
&= -\sum_i^K y_i^n \cdot \frac{\partial ln(\hat{y}_i)}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_k} \\
&= -\sum_i^K y_i^n \cdot \frac{1}{\hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_k} \\
&= -\left[\sum_{i \neq k}^K y_i^n \cdot \frac{1}{\hat{y}_i} \cdot (-\hat{y}_i \hat{y}_k)\right] - y_k \frac{1}{\hat{y}_k} \hat{y}_k (1 - \hat{y}_k) \\
&= -\left[\sum_{i \neq k}^K y_i^n \cdot (-\hat{y}_k)\right] - y_k (1 - \hat{y}_k) \\
&= \left[(\sum_i^K y_i^n \hat{y}_k) - y_k \hat{y}_k\right] - y_k + y_k \hat{y}_k \\
&= \left[\hat{y}_k (\sum_i^K y_i^n)\right] - y_k \\
&= -(y_k - \hat{y}_k)
\end{aligned}
$$

$$
\frac{\partial z_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_i^I w_{ki} x_j = x_j
$$

Thus, the gradient of the loss function w.r.t. the weights is:

$$
\begin{aligned}
\frac{\partial C(w)}{\partial w_{kj}} &= \frac{\partial C(w)}{\partial z_k} \frac{\partial z_k}{\partial w_{kj}} \\
&= -(y_k - \hat{y}_k) x_j
\end{aligned}
$$

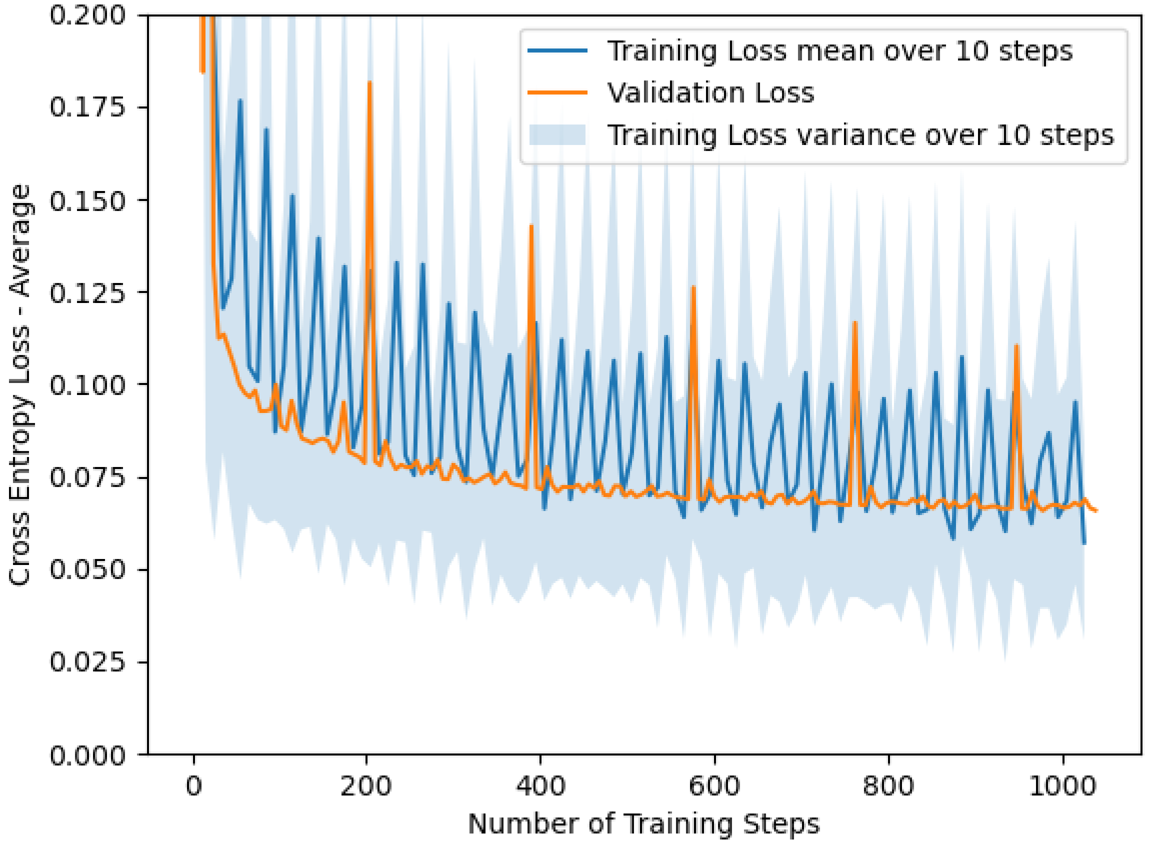
Task 2

Task 2a)

The functionality was implemented in `task2a.py` .

Task 2b)

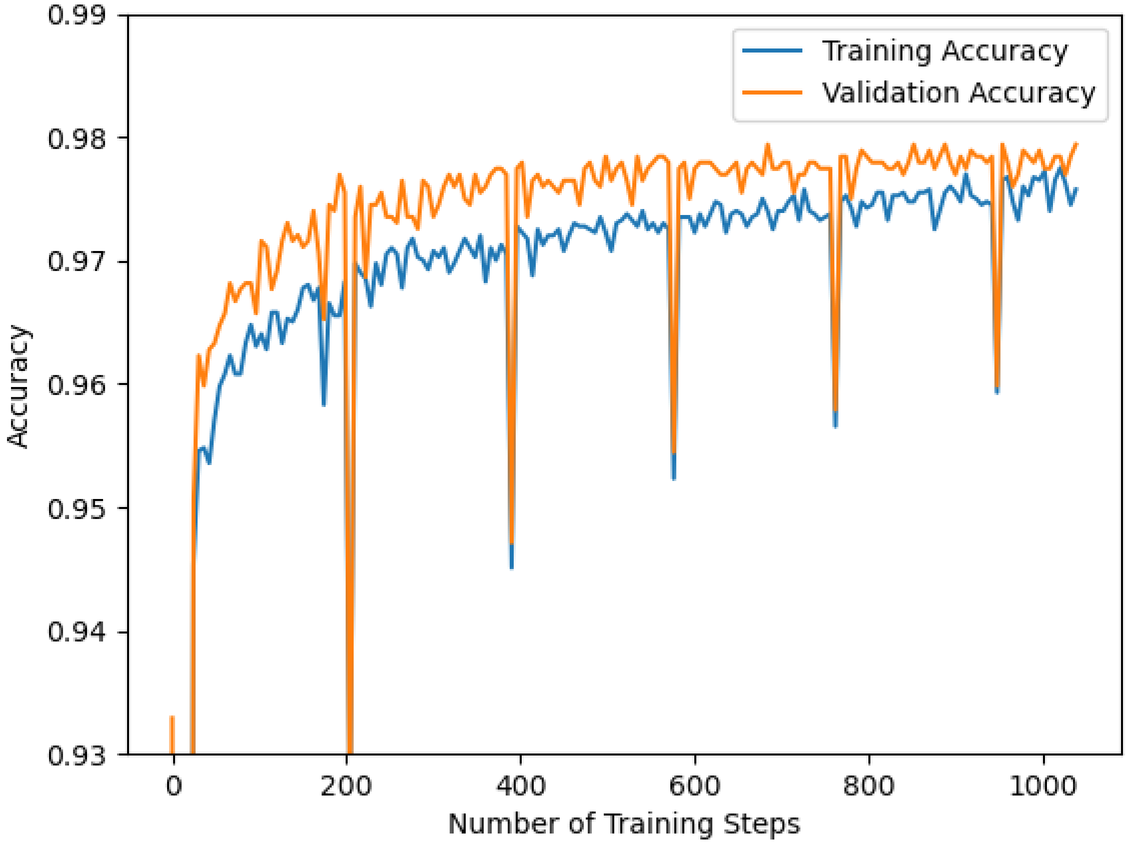
Logistic regression with mini-batch descent was for a single-layer neural network was implemented in `task2.py` .



The figure shows the loss of the binary classifier without shuffled data.

Task 2c)

The function `calculate accuracy` was implemented in `task2.py` .



The figure shows the accuracy of the binary classifier without shuffled data.

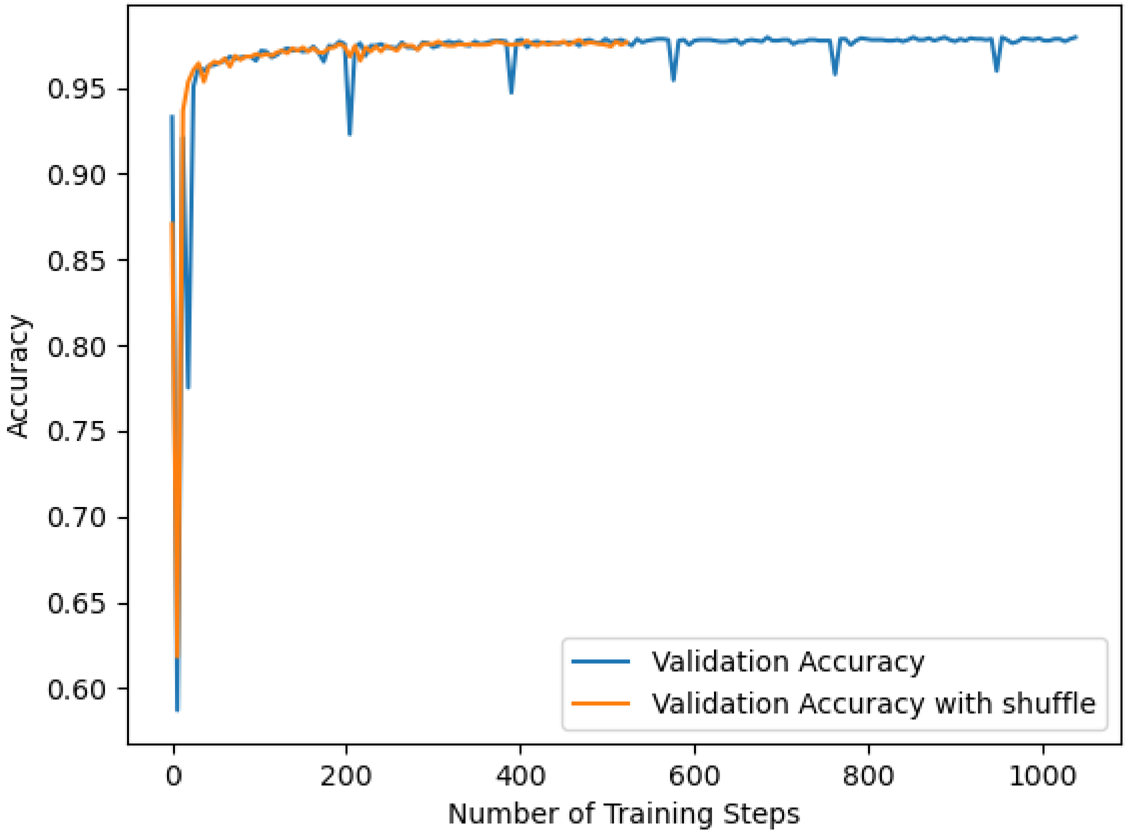
Task 2d)

Early stopping was implemented in the training loop in `trainer.py` .

Early stopping kicks in after 16 epochs. As we see in the graph, this is about halfway through the training.

Task 2e)

Dataset shuffling was implemented in `batch_loader` in `utils.py` .



Shuffling the data may help the training avoid the ordering of the training data against consistently affecting the performance of our classifier. For instance, if one part of the training data is very similar (or very different) from the validation data, we may expect the algorithm to perform better (or worse) after just optimizing the classifier to this data, in a sense overfitting to a part of the data. It seems that shuffling the data at every epoch reduces the variance of both training and validation accuracy.

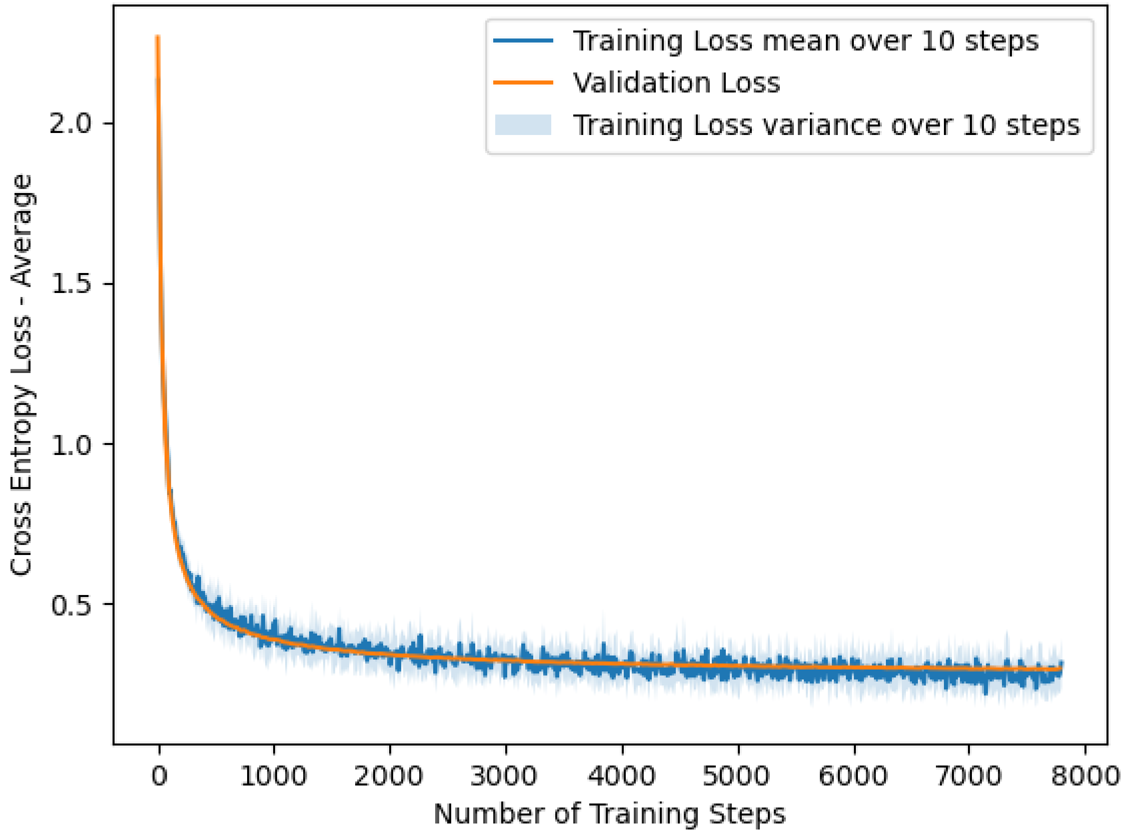
Task 3

Task 3a)

The functionality was implemented in `task3a.py` .

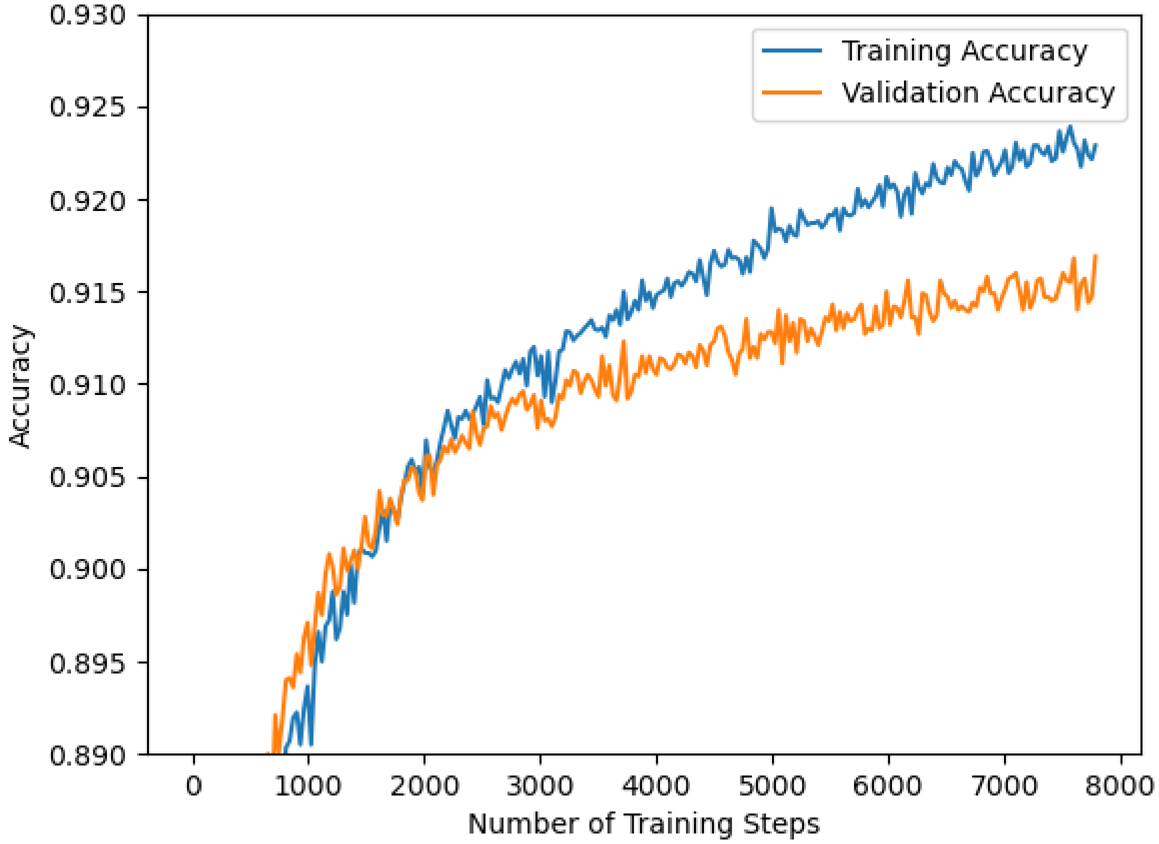
Task 3b)

Softmax regression was implemented in `task3.py` . The multiclass classifier was trained on the data. The data is shuffled at each epoch.



Task 3c)

A function for calculating the accuracy on multiple classes was implemented. The plot shows the training and validation accuracy over training.



Task 3d)

We see slight signs of overfitting, as we see that after about 2000 to 3000 training steps the accuracy on the validation data really starts to flatten out while the accuracy on the training data continues increasing.

Task 4

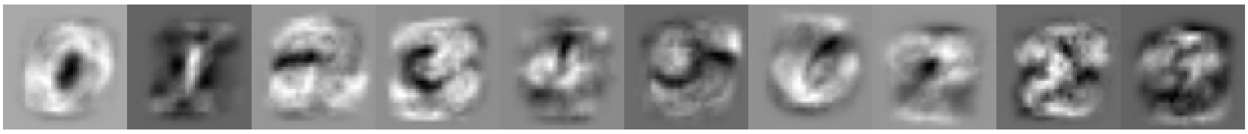
Task 4a)

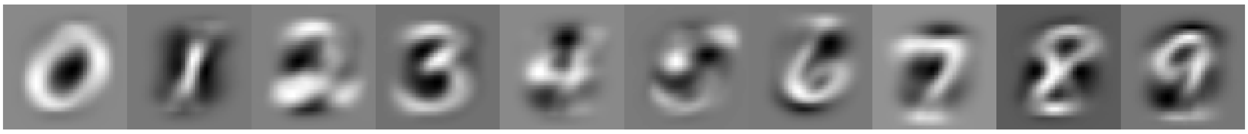
$$\begin{aligned}\frac{\partial J}{\partial w} &= \frac{\partial C(w)}{\partial w} + \lambda\frac{\partial R(w)}{\partial w}\\ \frac{\partial R}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{2}||w||^2 = \frac{\partial}{\partial w}w^\intercal w = w^\intercal\\ \frac{\partial J}{\partial w} &= -(y_k - \hat{y}_k)x + w\end{aligned}$$

We point out that this is a slight abuse of notation, because usually $\frac{\partial J}{\partial w}$ would be a row vector.

Task 4b)

L2 Regularization was implemented, the resulting weights after training can be seen plotted below.



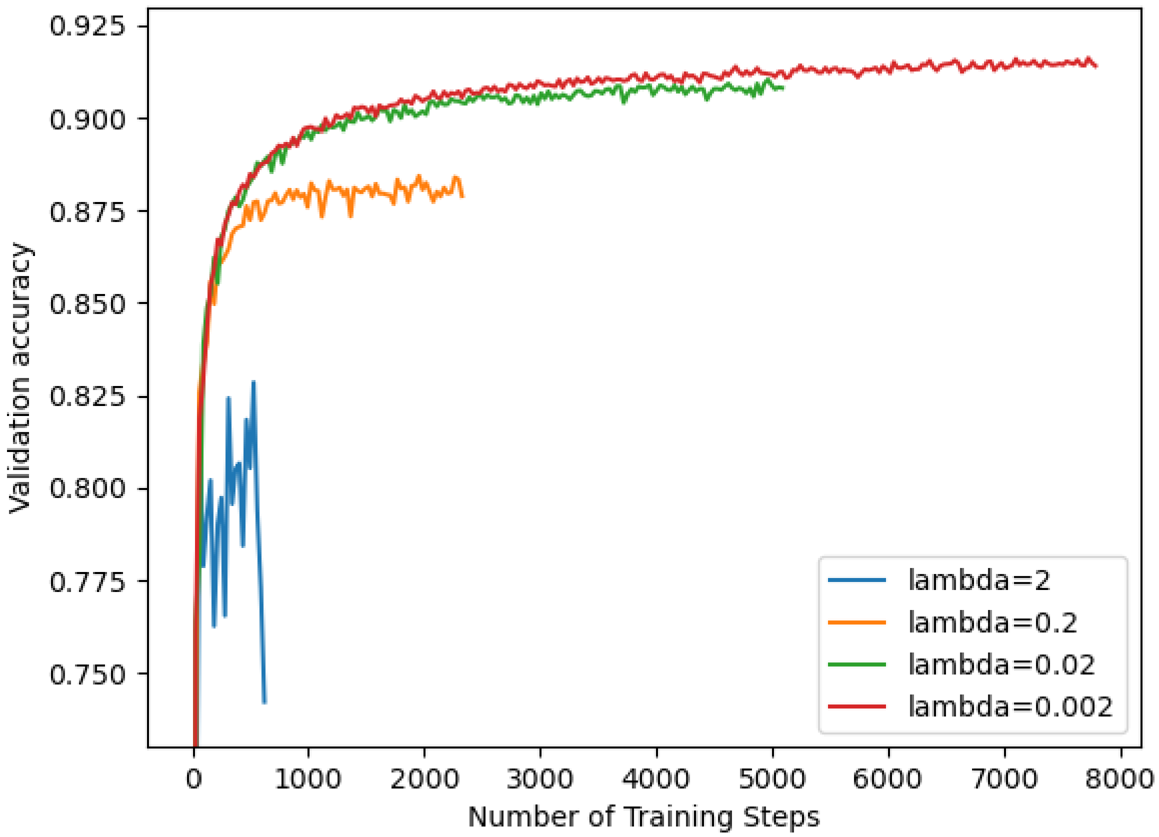


The weights are less noisy as there is now a cost to larger weights. An explantion for why this may lead to less noisy weights is that the weight on a pixel that may have little information value does not contribute that much to decreasing the loss on the dataset at large is attenuated - giving less noise.

Task 4c)

We see that the validation accuracy degrades with regularization. In this case, there is therefore a trade-off between denoising the network weights through regularization and validation accuracy.

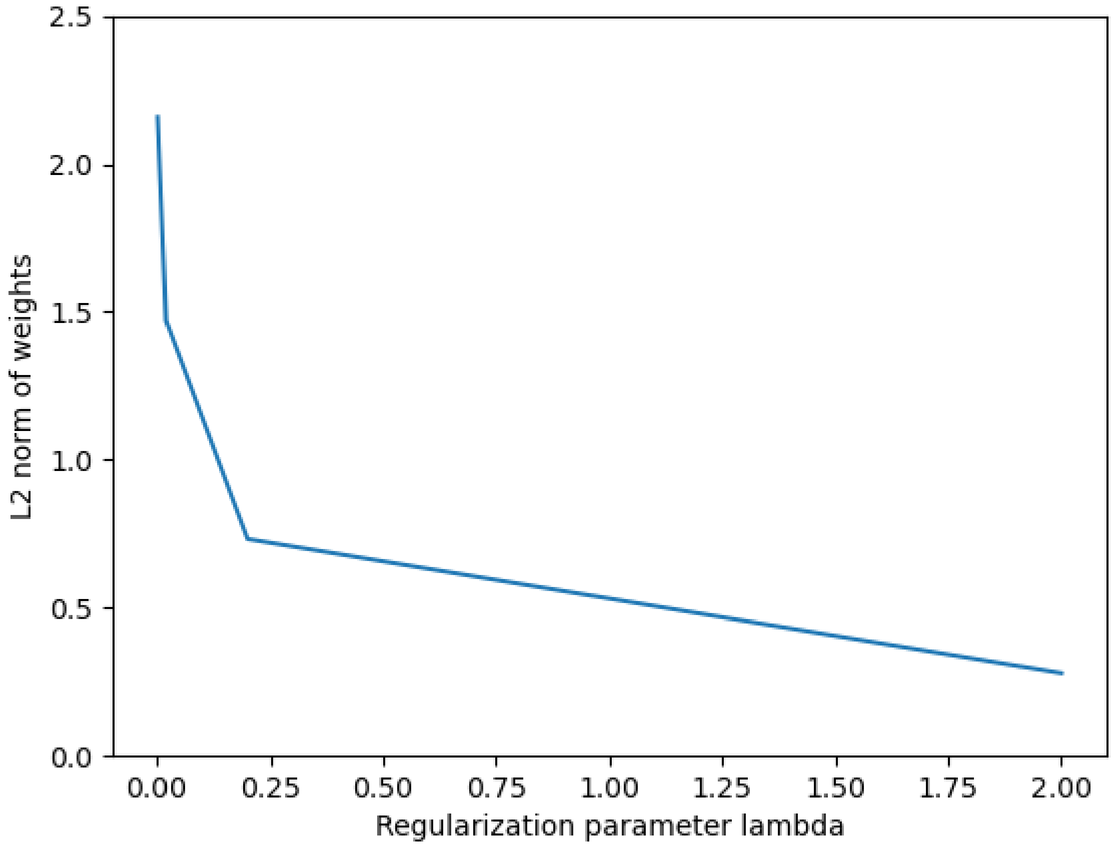
One reason may be that in this case, the "noisy" weights may be better for distinguishing numbers that have spatial perturbations (e.g. they may be shifted horizontally or vertically, or have different curves) even though the regularized weights look nicer to the human eye. Some classifiers like convolutional neural networks with max pooling layers may circumvent this, but our network does not.



Task 4d)

It seems that because any amount of regularization imposes a cost on weights that may have contributed to the predictive power of the classifier. It may be that the early stopping and simple classifier parameterization makes the regularization less necessary.

Task 4e)



We see that the norm of the network weights decreases when the regularization parameter lambda increases. This is expected, as the cost of larger network weights increases with increasing lambda.