Assignment 1 Report

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)

$$C(\omega) = \frac{1}{N}\sum_{n=1}^{N} C^N(\omega) = \frac{1}{N}\sum_{n=1}^{N}[-(y^n ln(\hat{y}^n) + (1-y^n)ln(1-\hat{y}^n))]$$

The derivative of C is easy to compute when we have found the derivative of C^n.

$$\frac{\partial C^n}{\partial \omega_i} = \frac{\partial C^n(\omega)}{\partial \hat{y}^n}\frac{\partial \hat{y}^n}{\partial \omega_i} = [-y^n * \frac{1}{\hat{y}^n}\frac{\partial \hat{y}^n}{\partial \omega_i} - (1-y^n)\frac{1}{1-\hat{y}^n}(-1)\frac{\partial \hat{y}^n}{\partial \omega_i}]$$

We have that $\hat{y}^n = f(x_i^n)$

$$\frac{\partial C^n}{\partial \omega_i} = [-\frac{y^n}{\hat{y}^n}\frac{\partial f(x_i^n)}{\partial \omega_i} + \frac{1-y^n}{1-\hat{y}^n}\frac{\partial f(x_i^n)}{\partial \omega_i}]$$

The derivative of f with regards to ω is: $\frac{\partial f^{x_i^n}}{\partial \omega_i} = x_i^n f(x_i^n)(1-f(x_i^n))$

$$\frac{\partial C^n}{\partial \omega_i} = [-\frac{y^n}{\hat{y}^n}x_i^n f(x_i^n)(1-f(x_i^n)) + \frac{1-y^n}{1-\hat{y}^n}x_i^n f(x_i^n)(1-f(x_i^n))]$$

$$\frac{\partial C^n}{\partial \omega_i} = [-y^n x_i^n(1-\hat{y}^n) + (1-y^n)x_i^n \hat{y}^n]$$

This yields the resulting derivative:

$$\frac{\partial C^n}{\partial \omega_i} = (\hat{y}^n - y^n)x_i^n$$

task 1b)

The cross entropy function:

$$C(\omega_{kj}) = \frac{1}{N}\sum_{n=1}^{N} C^N(\omega_{kj}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}[y_k^n ln(\hat{y}_k^n)]$$

We now have a multiclass detector such that the outputs of our neural network and the corresponding targets are also defined by the number of output nodes, K.

Again we find the derivative of C by finding the derivative of C^n.

We have that $\hat{y}_k^n = \frac{e^{z_k}}{\sum_{k'=1}^{K'} e^{z_{k'}}}$

$$C^n(\omega_{kj}) = -\sum_{k=1}^{K}[y_k^n ln(\frac{e^{z_k}}{\sum_{k'=1}^{K'} e^{z_{k'}}})]$$

$$\frac{\partial C^n}{\partial \omega_{kj}} = \frac{\partial C^n(\omega_{kj})}{\partial z_k}\frac{\partial z_k}{\partial \omega_{kj}}$$

The derivative of z_k with respect to ω_{kj} is:

$$\frac{\partial z_k}{\partial \omega_{kj}} = x$$

For the derivative of C^n with respect to z_k we have to consider two cases. When k=k' and when k!=k'. First we use the core rule:

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = -\sum_{k=1}^{K}[y_k^n \frac{u}{*} u').$$

where $u = \frac{e^{z_k}}{\sum_{k'=1}^{K'} e^{z_{k'}}}$ and $u' = \frac{(e^{z_k})' \sum_{k'=1}^{K'} e^{z_{k'}} - e^{z_k}(\sum_{k'=1}^{K'} e^{z_{k'}})'}{(\sum_{k'=1}^{K'} e^{z_{k'}})^2}$

Case 1: k=k'

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = y_{k'}^n \frac{\sum_{k'=1}^{K'} e^{z_{k'}}}{e^{z_k}}[\frac{e^{z_k} * \sum_{k'=1}^{K'} e^{z_{k'}} - e^{z_k}(\sum_{k'=1}^{K'} e^{z_{k'}})'}{(e^{z_{k'}})^2}]$$

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = y_{k'}^n \frac{1}{\hat{y}_k^n}[\hat{y}_k - \hat{y}_k \hat{y}_{k'}] = y_{k'}^n(1 - \hat{y}_{k'})$$

Case 2: k≠k'

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = y_{k'}^n \frac{\sum_{k'=1}^{K'} e^{z_{k'}}}{e^{z_k}}[\frac{0 * \sum_{k'=1}^{K'} e^{z_{k'}} - e^{z_k}(\sum_{k'=1}^{K'} e^{z_k})'}{(e^{z_{k'}})^2}]$$

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = y_k^n \frac{1}{\hat{y}_k^n}[-\hat{y}_k \hat{y}_{k'}] = -y_k^n \hat{y}_{k'}$$

This gives the final expression:

$$\frac{\partial C^n(\omega_{kj})}{\partial z_k} = x[y_{k'}^n(1 - \hat{y}_{k'}) + (-y_k^n(\hat{y}_{k'}))] = x[y_{k'}^n - y_{k'}^n \hat{y}_{k'} - y_k^n \hat{y}_{k'}]$$

We did not managed to get the correct final expression and did not have time to find the error.

Task 2

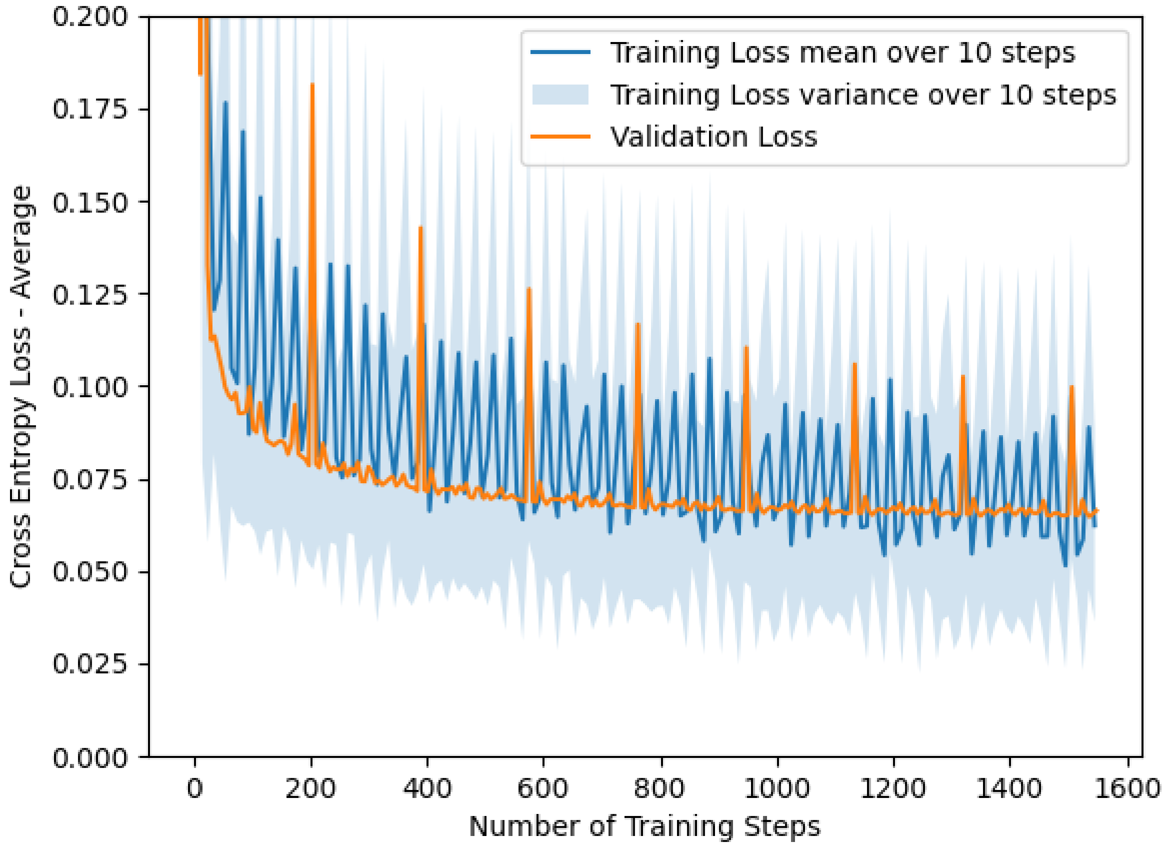
Task 2a)

The functions was implemented in VSCode successfully.

Task 2b)

The functionality of completing the logistic regression with mini-batch gradient descent for a single layer was implemented. First a forward pass, then the loss was calculated based on the output compared to the expected target output. Then backpropegation was computed and the weights was updated based on the calculated gradient.

The resulting loss over training steps is visualized in the figure below. This is before we have implemented early stoppage, therefore the NN train till there are no more samples.

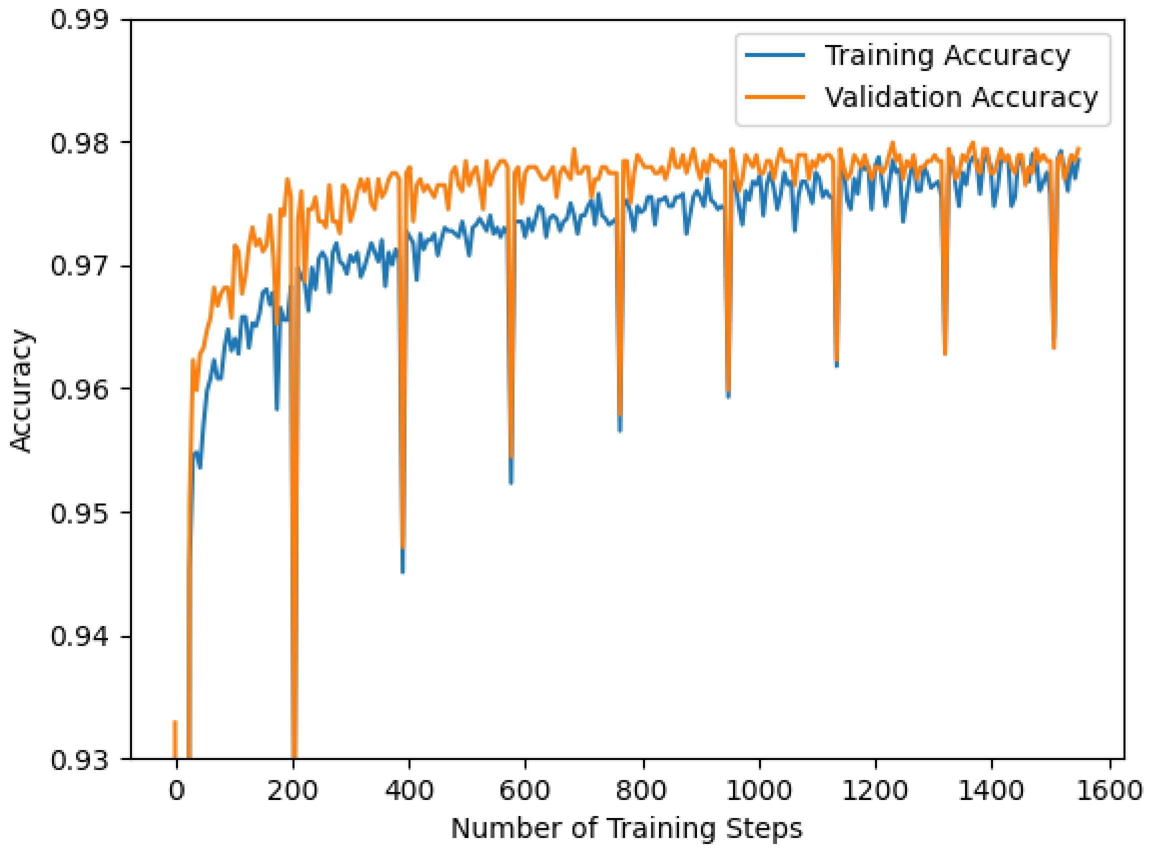


The loss starts to stagnate around 600 sample with very small improvement after that. The final loss is around 0.075.

During the training the loss have some significant peaks which are large compared to the pattern. This originates from that periodically a very hard test set comes along where it misses completely. This is periodically because all the bad test set come at the same time resulting in a very large peek. If the samples were to be shuffled around in the dataset then the peaks would disappear.

Task 2c)

The accuracy function was implemented successfully in the code. The resulting accuracy over training is visualized in the graph below.



The Validation accuracy shows to reach up to 98 percentage which is really good. We also see thath is stagnates up to 600 training steps similar to the loss graph which is resonable. Also it has the same peaks as for the loss graph.

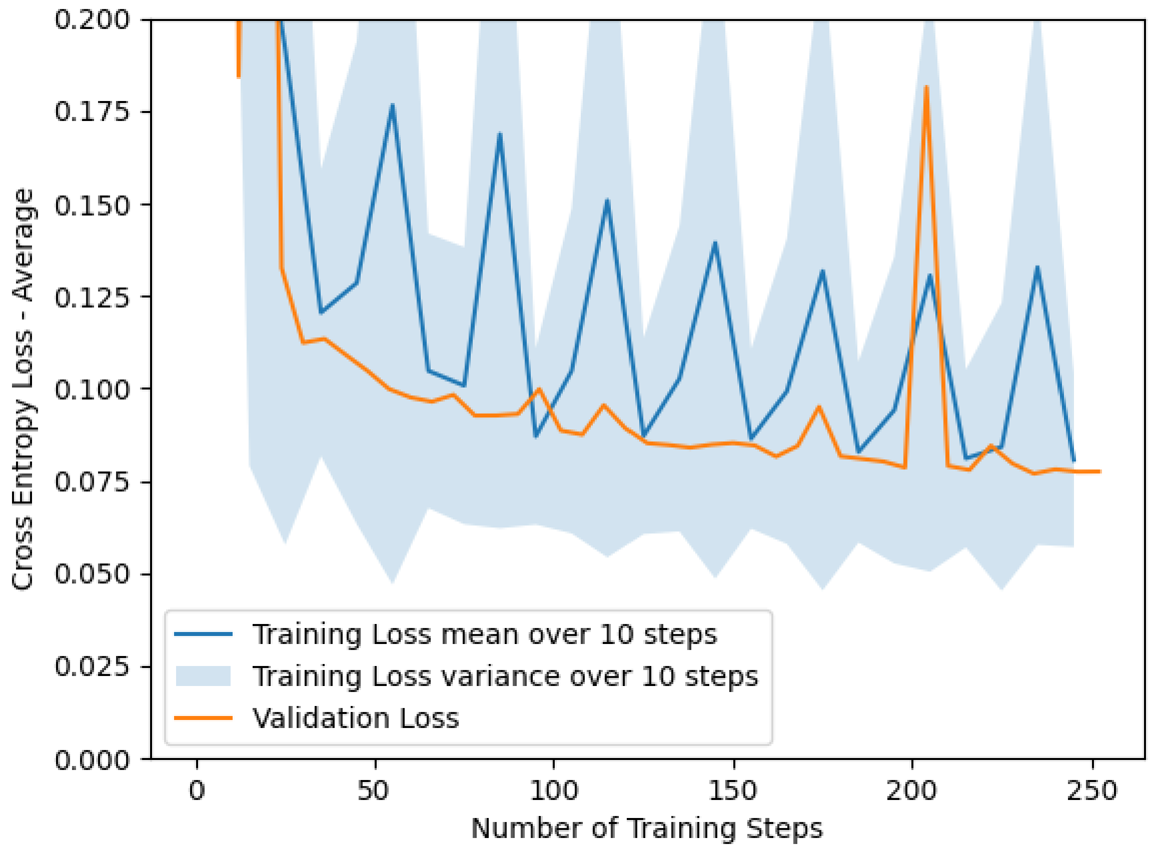
The validation set is a set used to test the NN, while the training set is a different dataset used to train and tune the NN. During training the validation set has a larger accuracy than the training set. This is weird as one would expect that the NN would do better on the dataset it is training on. However, this could be completely random and we see that at the end the training set has equal or slitghly better accuracy than the validation set.

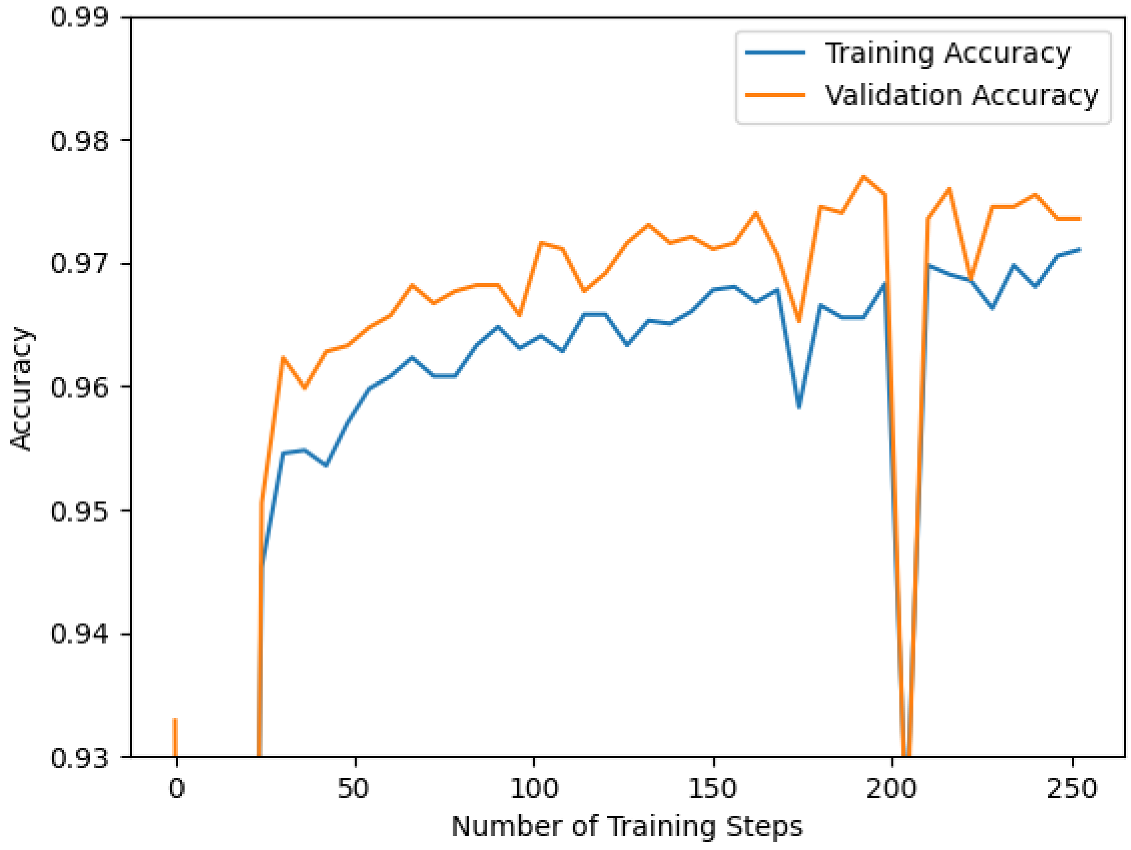
There is no overfitting for this NN.

Task 2d)

The early stoppage was implemented in the trainer.py. It was confusing how to implement this, however a reasonable method was found. At each validation, 20% of training set, we count if the new accuracy was less than the old accuracy. If this counter reaches 10 then we will stopp early. This means that after 10 checks the accuracy had not improved. This provided a much better plot which stops at a reasonable time. Instead of running for 1600 samples we rather run for closer to 300 steps. We could increase the counter to get a slightly better accuracy. However, the gain in accuracy might not be worth the extra computations.

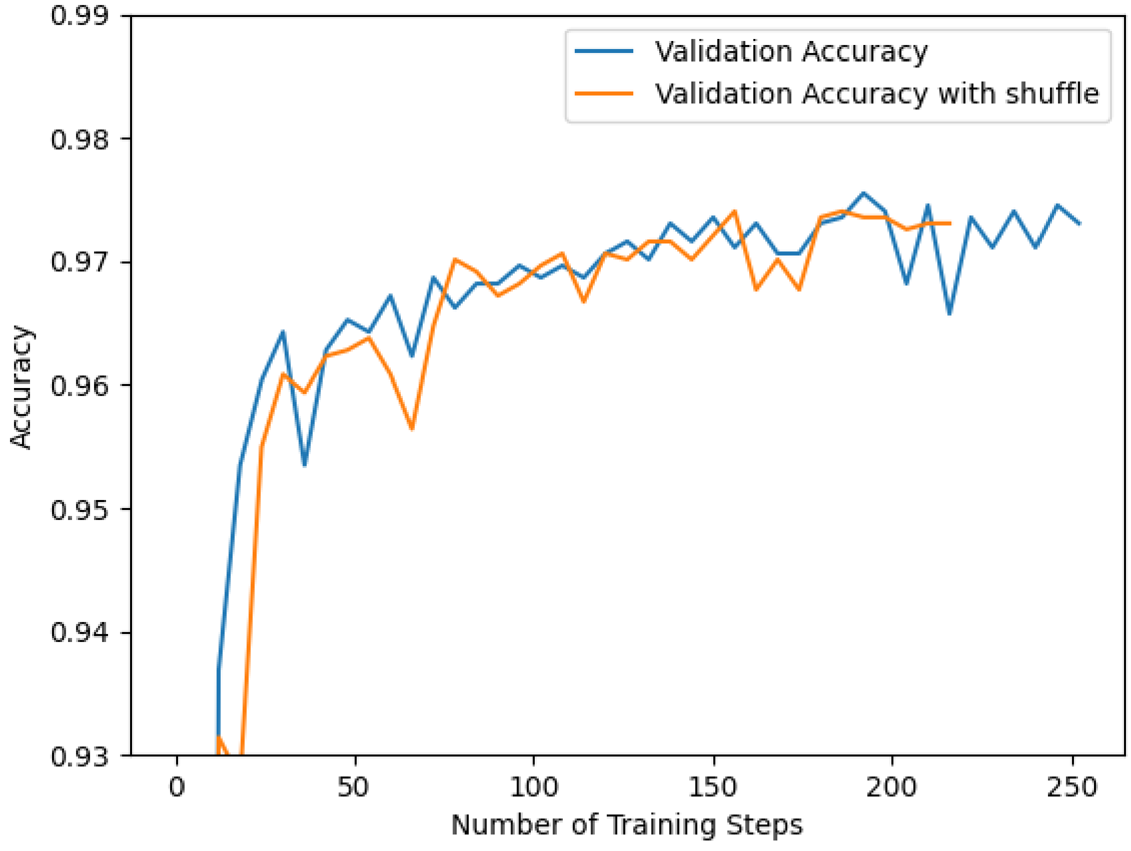
The resulting results are visualized in the graphs below.

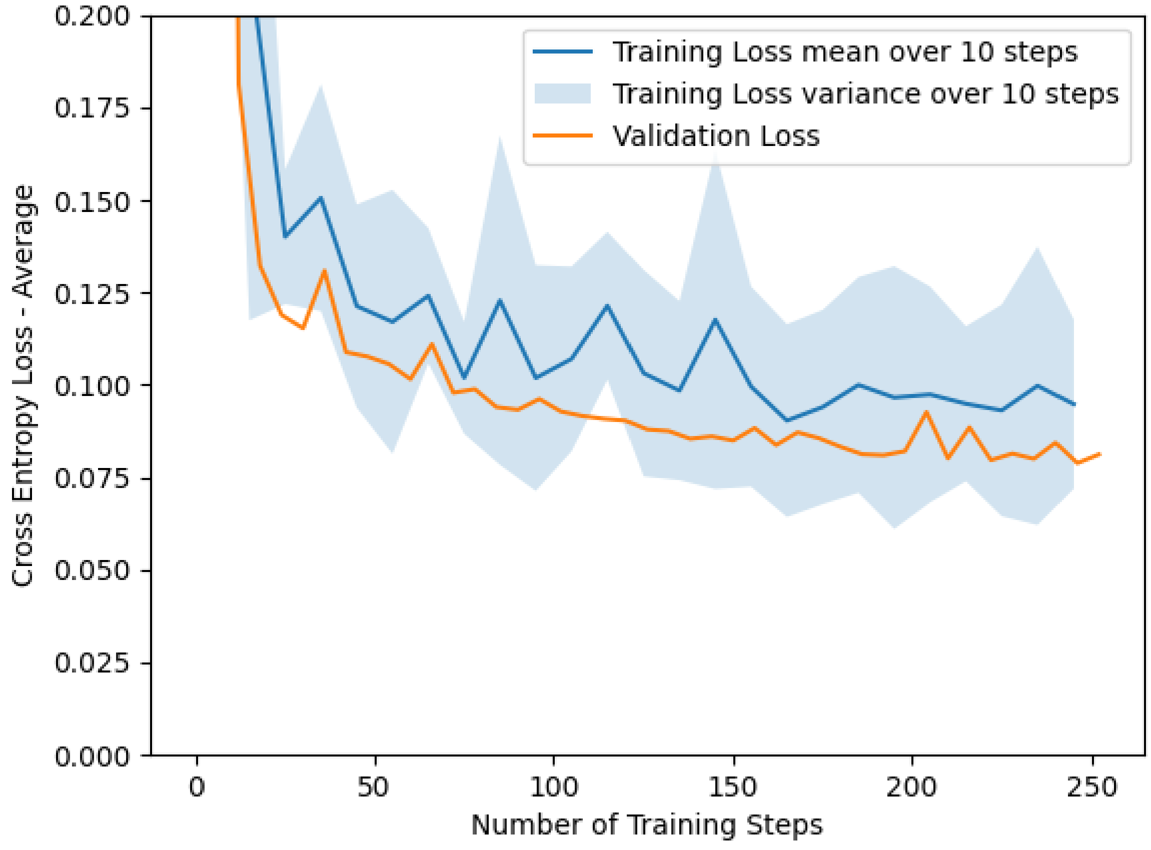

Cross Entropy Loss - Average
Number of Training Steps
Training Loss mean over 10 steps
Training Loss variance over 10 steps
Validation Loss

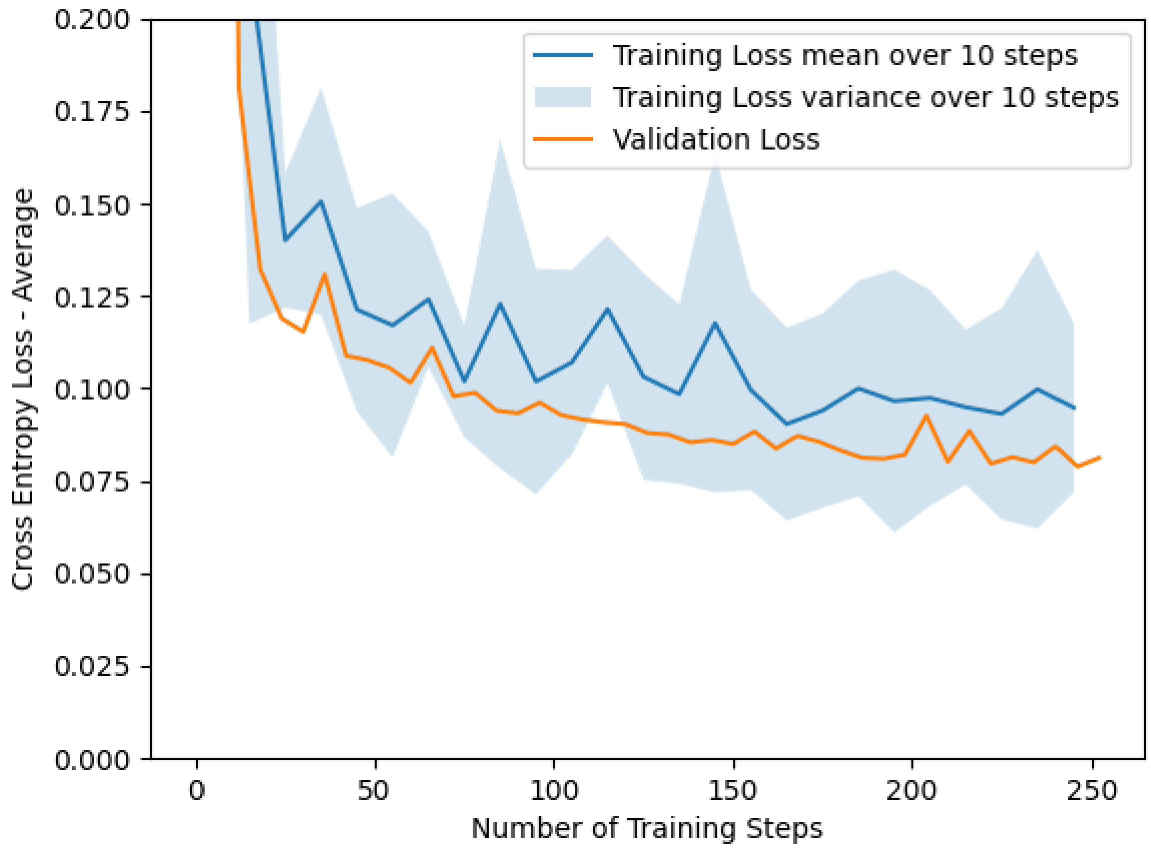

Accuracy
Number of Training Steps
Training Accuracy
Validation Accuracy

Task 2e)

After implementing a shuffle functionality the resulting progress over the training steps have removed the peaks. This is depicted below.







The peaks disapear because the bad dataset that came periodically is now random in the datasets. That means that the bad examples are spread out reducing peaks.

The first plot visualize accuracy with and without shuffling. However, it looks like there both are shuffled. So there might be something wrong with the code that compute the data that is plotted.

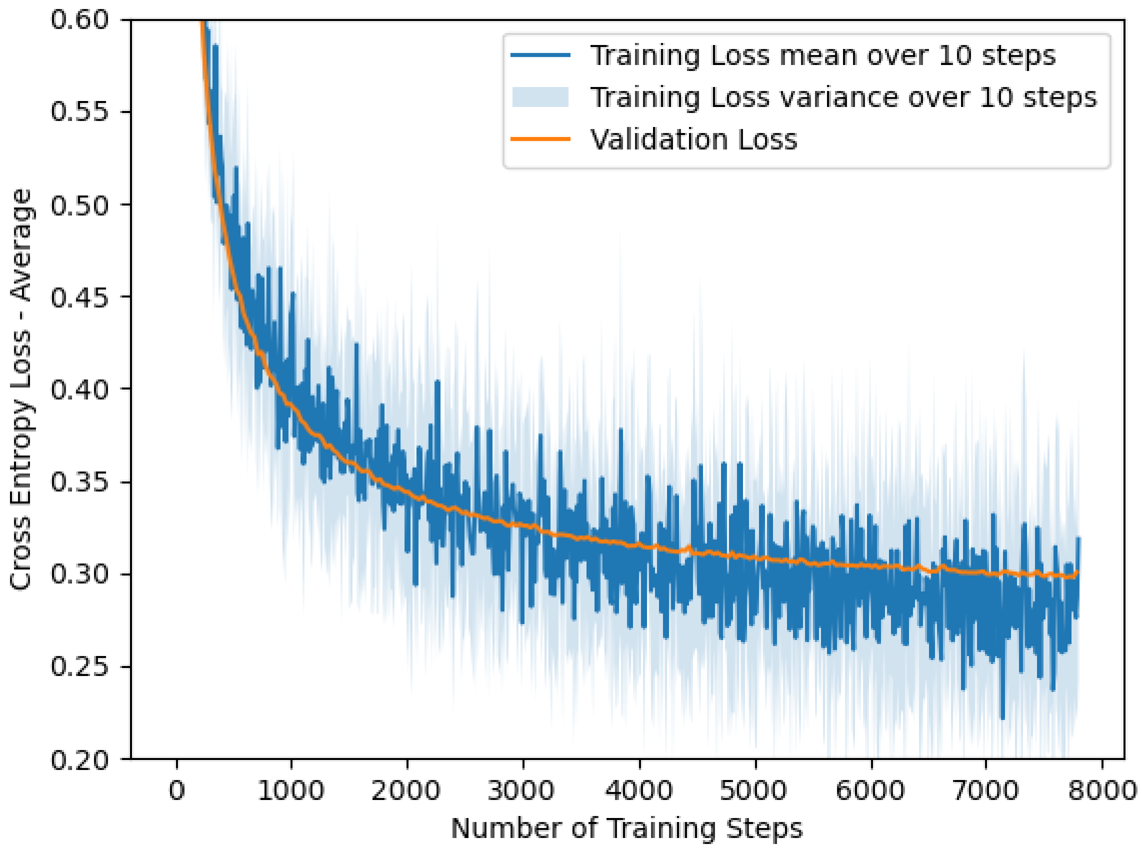
Task 3

Task 3a)

The functions were implementing the the code sucessfully, with no errors.

Task 3b)

The code was implemented similarly to Task 2b). The resulting loss over traning steps are visualized in the graph below.

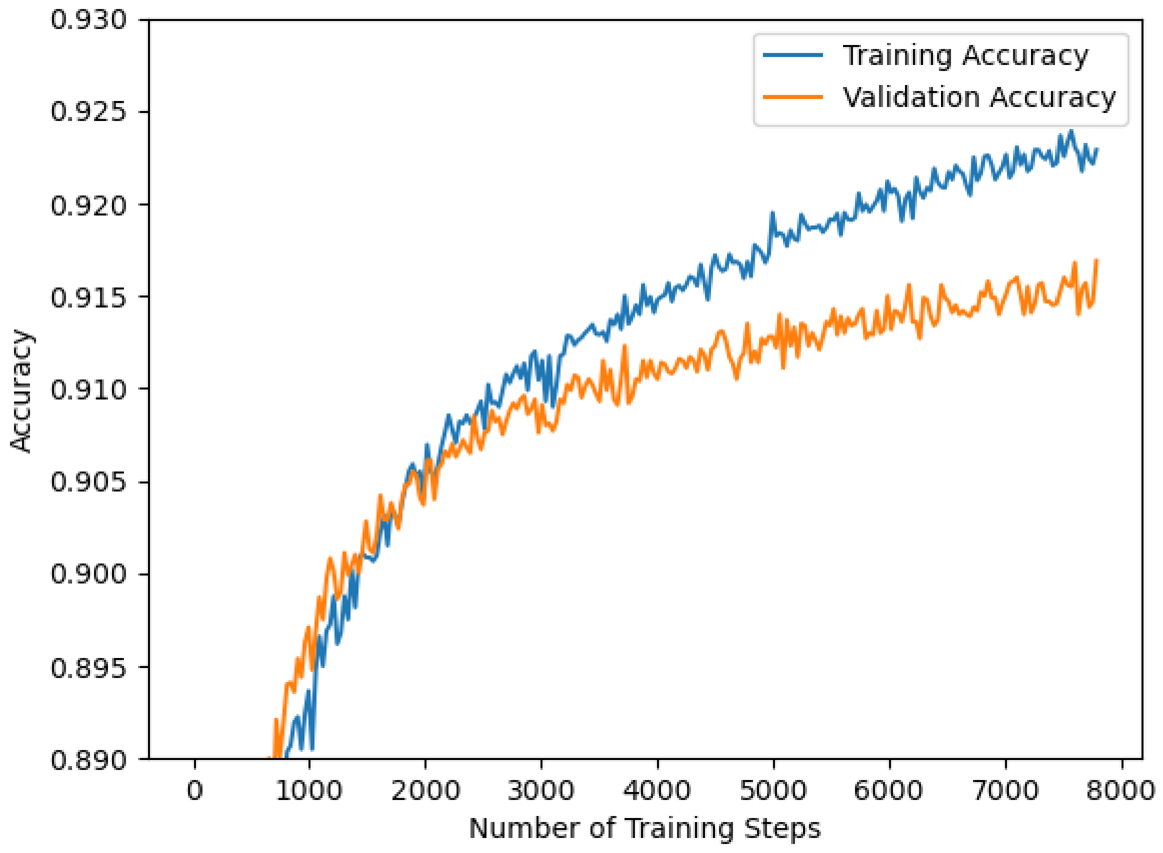


The loss has a slightly less decrease over time such that the early stoppage dosen't kick in before 3000 steps. It is also expected for it to take longer to achieve a good loss value compared to the task 2). This is because we now have a classification problem of 10 output nodes instead of a binary problem of 1 output node.

There are no peaks in this problem as the shuffling of the dataset is activated.

Task 3c)

The function for the accuracy of the NN was implemented successfully. The resulting accuracy over training steps is visualized in graph below.



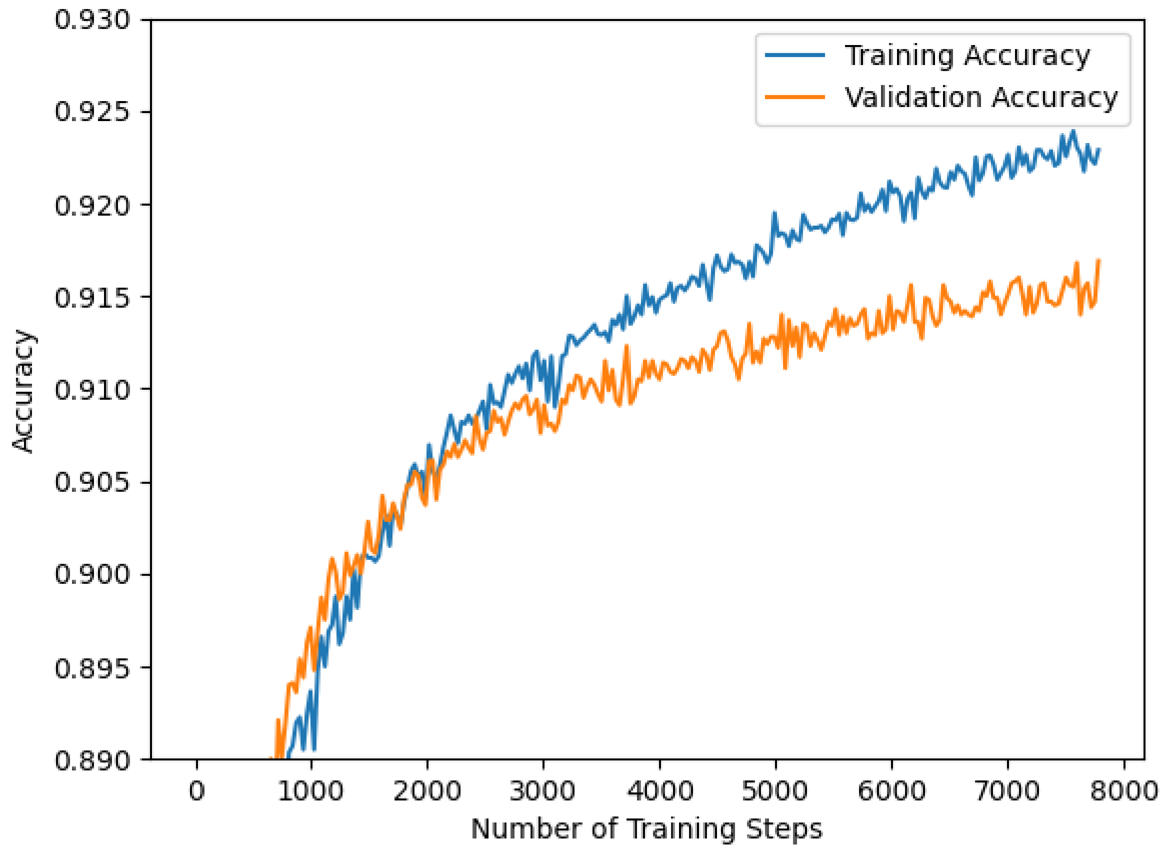
The accuracy acheive up to 91 percentage. This is considerably lower than 98 percent in task 2b), but also expected as the complexity of the problem is increased.

We see again that the validation accuracy is larger for the first half similarly to task 2). At the end the training accuracy is better than the validation which is expected. As it should be better at the dataset it trains on than the dataset it is tested on.

Task 3d)

Do we observe any overfitting from the graph in problem 3c)?

Also here we have no overfitting. Overfitting could be found when the NN does considerably much better on the training set than the validation set. This means that the NN is an expert on the specific dataset it has trained on, but does porely on the dataset it is tested on. We could see some overfitting at the end of the accuracy plot but the early stoppage ensure that overfitting is not a big problem. If we were to remove the early stoppage and let the NN train on the entire dataset overfitting is expected to become a problem. This is shown in the figure below:



The accuracy improved a lot from the early stoppage, but this is only overfitting.

Task 4

Task 4a)

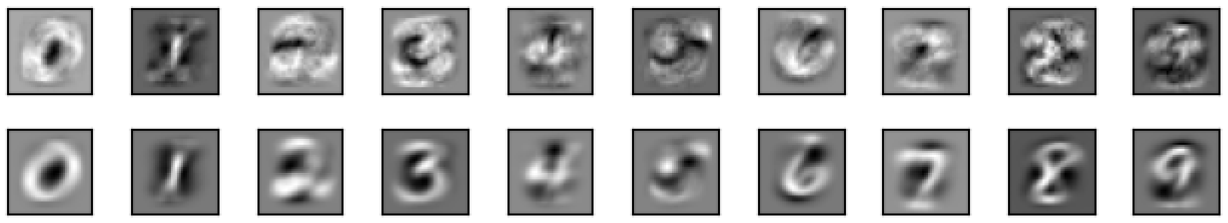
$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial C(w)}{\partial w_{i,j}} + \lambda \frac{\partial R(w)}{\partial w_{i,j}}$

$\frac{\partial C(w)}{\partial w_{i,j}} = -\frac{1}{N} x_j^n (y_i^n - \hat{y}_i^n)$

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

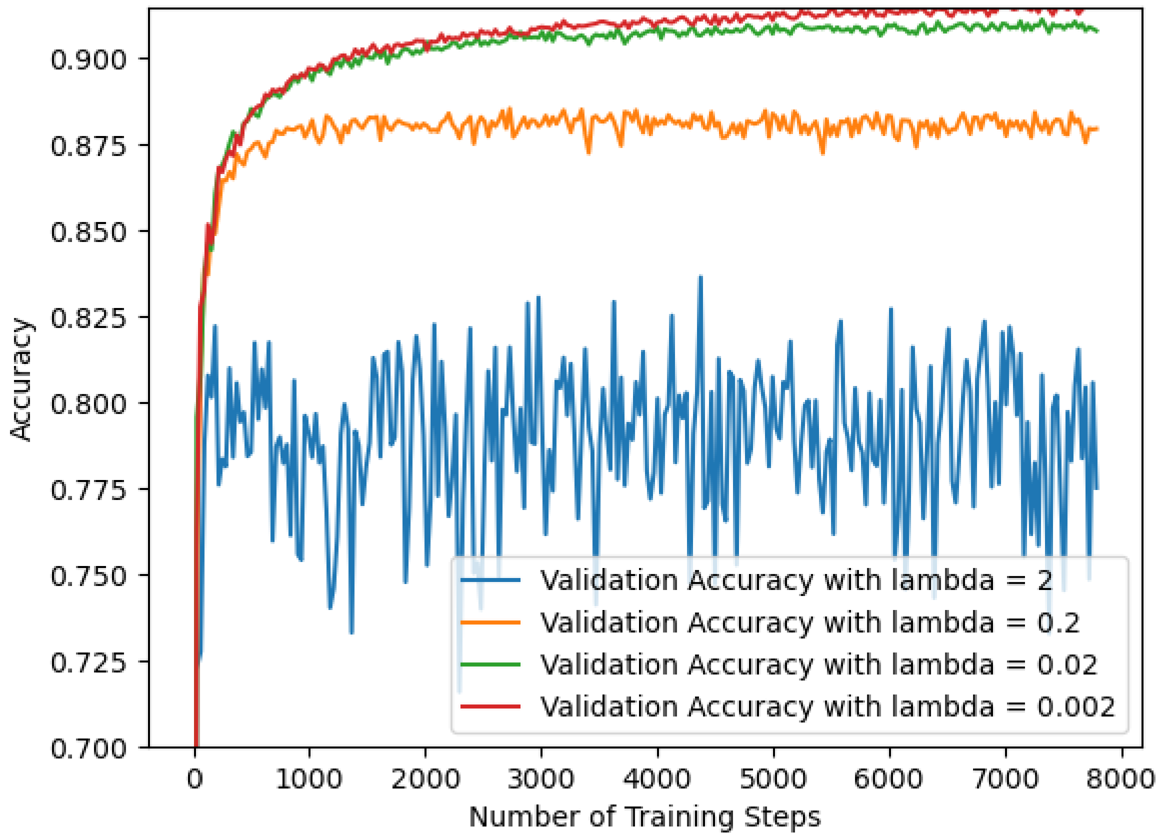
$\frac{\partial J}{\partial w_{i,j}} = -\frac{1}{N} x_j^n (y_i^n - \hat{y}_i^n) + \lambda w_{i,j}$

Task 4b)



The images represent the weights for each digits for two Neural Networks trained respectively with λ equals to 0.0 and 2.0. The weights for the model with higher λ are less noisy because of the regularization, it reduces the high difference of intensity between pixel that are close to each other.

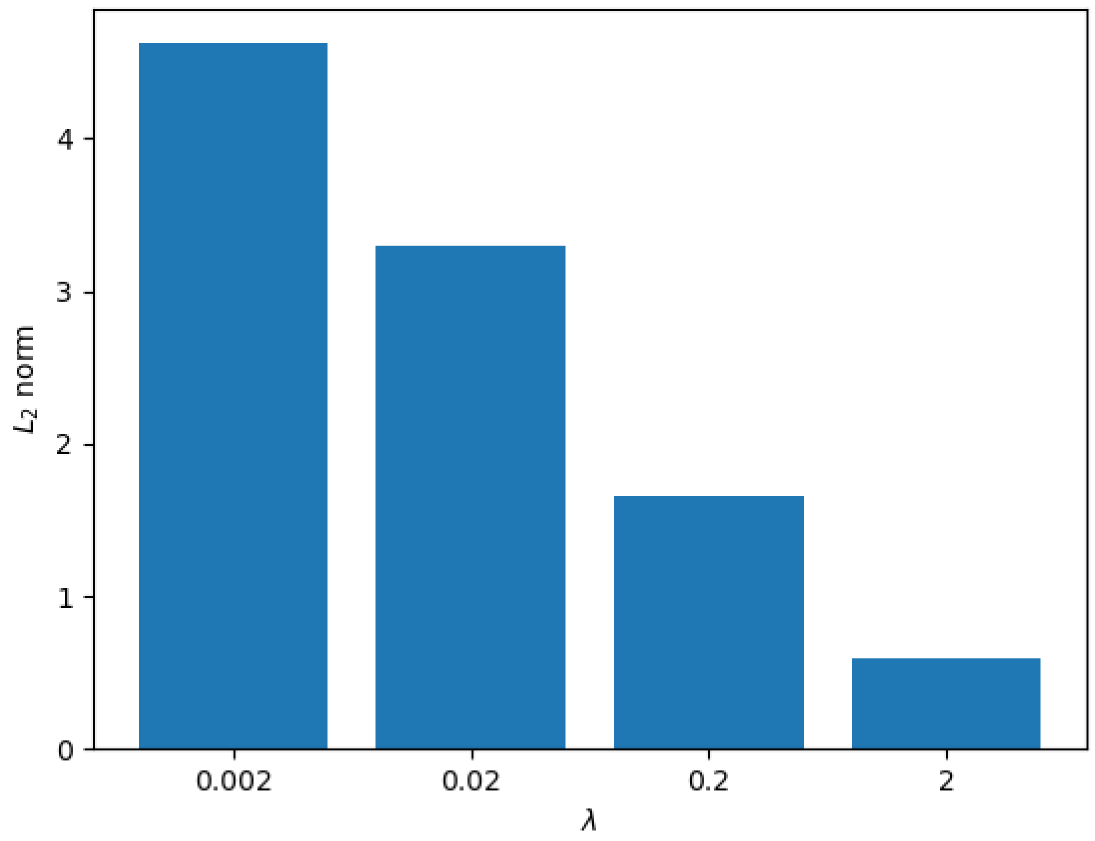
Task 4c)



Task 4d)

The reason for this phonomenon is hidden in the semplicity of our model, even without regularization we obtain an optimal result without overfitting, which is one of the major reason for using regularization. Probably the method gives some advantages in situations where the Neural Network is much more complicated and so overfitting could be an issue.

Task 4e)



It is evident that when λ is increased there is a decrease of the L_2 norm, which is caused by the regularization process.