NTNU

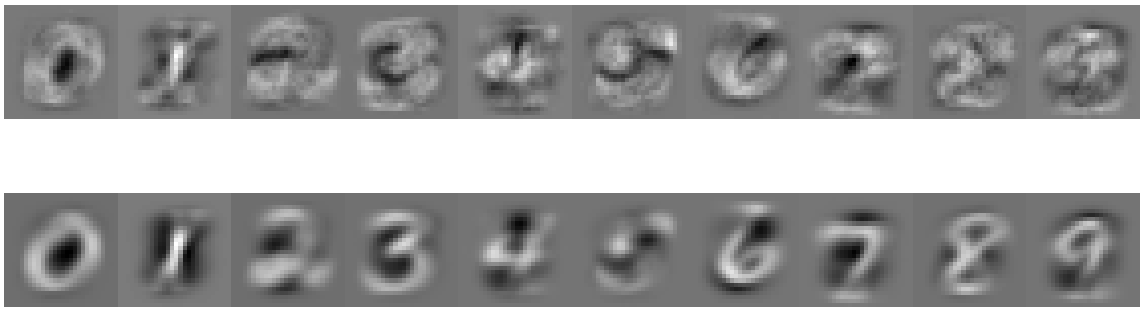
Kunnskap for en bedre verden

Department of Computer Science

TDT4265 - Computer Vision and Deep Learning

Assignment 1

Kolbjørn Kelly
Sander Endresen



February 5^{th} 2021

Contents

1 Task 1: Theory

1.1 Task 1a: Gradient for Logistic Regression

The average cost over N data points is given by

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

where $C^n(w) = -(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1\text{-}\hat{y}^n))$. The output of the network, $\hat{y}$, is given by the sigmoid function

$$P(x \in C_1|x) = \hat{y} = f(x) = \frac{1}{1+e^{-w^T x}}, \tag{2}$$

and the labels, y, are provided by the training data. To obtain the gradient of C w.r.t. each weight, the chain rule must be applied. The chain rule yields

$$\frac{\partial C^n(w)}{\partial w_i} = \frac{\partial C^n(w)}{\partial \hat{y}^n}\frac{\partial \hat{y}^n}{\partial w_i}, \tag{3}$$

where

$$\frac{\partial C^n(w)}{\partial \hat{y}^n} = -y^n\frac{1}{\hat{y}^n} + (1-y^n)\frac{1}{1-\hat{y}^n}, \tag{4}$$

and

$$\frac{\partial \hat{y}^n}{\partial w_i} = x_i^n \hat{y}^n(1-\hat{y}^n). \tag{5}$$

Hence,

$$\begin{aligned}
\frac{\partial C^n(w)}{\partial w_i} &= (-y^n\frac{1}{\hat{y}^n} + (1-y^n)\frac{1}{1-\hat{y}^n})(x_i^n\hat{y}^n(1-\hat{y}^n)) && (6)\\
&= -x_i^n y^n(1-\hat{y}^n) + x_i^n\hat{y}^n(1-y^n) && (7)\\
&= -(y^n - y^n\hat{y}^n - \hat{y}^n + y^n\hat{y}^n)x_i^n && (8)\\
&= -(y^n - \hat{y}^n)x_i^n && (9)
\end{aligned}$$

1.2 Task 1b: Gradient for Softmax Regression

For multiple classes, the cross entropy cost function is defined as (1), but now with

$$C^n(w) = -\sum_{k=1}^{K} y_k^n ln(\hat{y}_k^n) \tag{10}$$

Here, $\hat{y_k}$ is the Softmax function, given by

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

where

$$z_k = w_k^T \cdot x = \sum_i^I w_{k,i} \cdot x_i$$

and

$$\sum_k^K \hat{y_k} = 1.$$

We want to minimize the multi-class cross entropy cost function. We rewrite it to obtain

$$C^n(w) = -\sum_{k=1}^{K} y_k^n ln\Big(\frac{e^{z_k}}{\sum_{k'}^K e^{z_{k'}}}\Big) \tag{11}$$

$$= -\sum_{k=1}^{K} y_k^n \left[ln(e^{z_k}) - ln\Big(\sum_{k'}^{K} e^{z_{k'}}\Big)\right] \tag{12}$$

$$= -\sum_{k=1}^{K} y_k^n z_k + \sum_{k=1}^{K} y_k^n ln\Big(\sum_{k'}^{K} e^{z_{k'}}\Big) \tag{13}$$

$$\tag{14}$$

Now we want to calculate the gradient w.r.t. the weights. We use the fact that $\sum_{k=1}^K y_k^n = 1$, and split into two cases where k' = k and k' $\neq$ k. When differentiating w.r.t w_{kj}, because $\sum_{k=1}^K y_k^n = 1$, the term $\sum_{k=1}^K y_k^n ln(\sum_{k'}^K e^{z_{k'}})$ will give one $\sum_{k'}^K e^{z_{k'}}$ term, so the y and k term will only pick out when k is equal to j, and $y_k^n x_j^n$ remains after differentiation. We then have the following:

$$\frac{\partial C(w)}{\partial w_{kj}} = -y_k^n x_j^n + \frac{1}{\sum_{k'}^K e^{z_{k'}}} e^{z_k} x_j^n \tag{15}$$

$$= \left(\frac{e^{z_k}}{\sum_{k'}^K e^{z_{k'}}} - y_k^n\right) x_j^n \tag{16}$$

$$= (\hat{y}_k^n - y_k^n) x_j^n \tag{17}$$

$$= -x_j^n (y_k^n - \hat{y}_k^n) \tag{18}$$

2 Task 2: Logistic Regression through Gradient Descent

2.1 Task 2a and 2b: Logistic Regression

A plot of the validation loss is shown in figure 1.

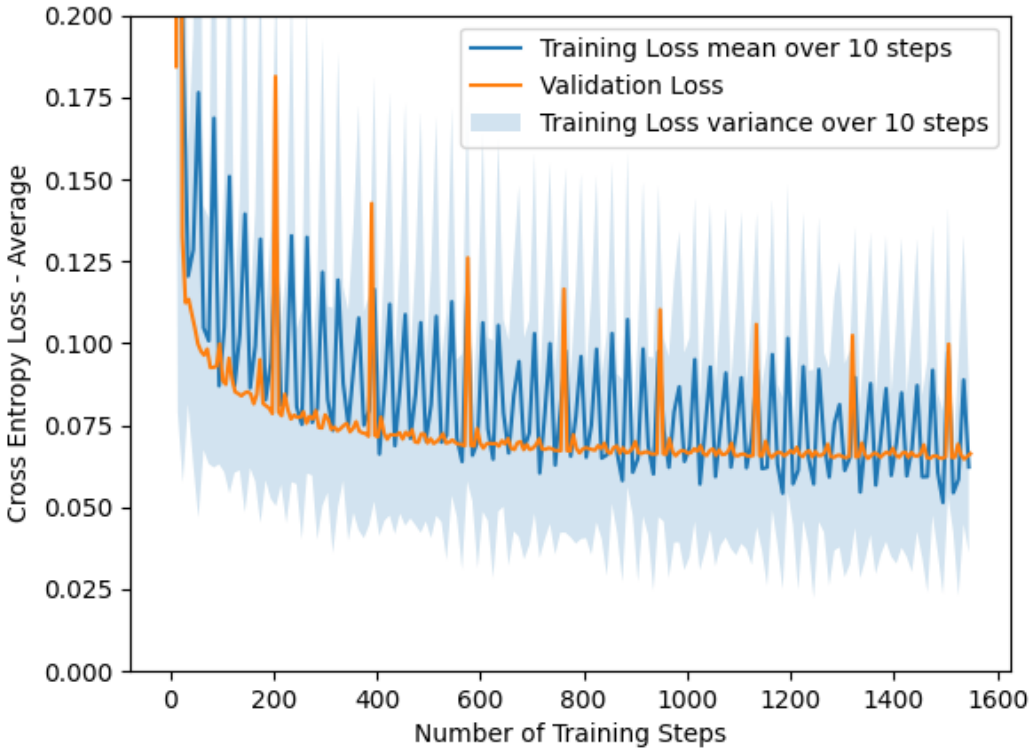


Figure 1: 2b) Validation Loss

2.2 Task 2c: Binary classification accuracy

A plot of the training- and validation accuracy is shown in figure 2.

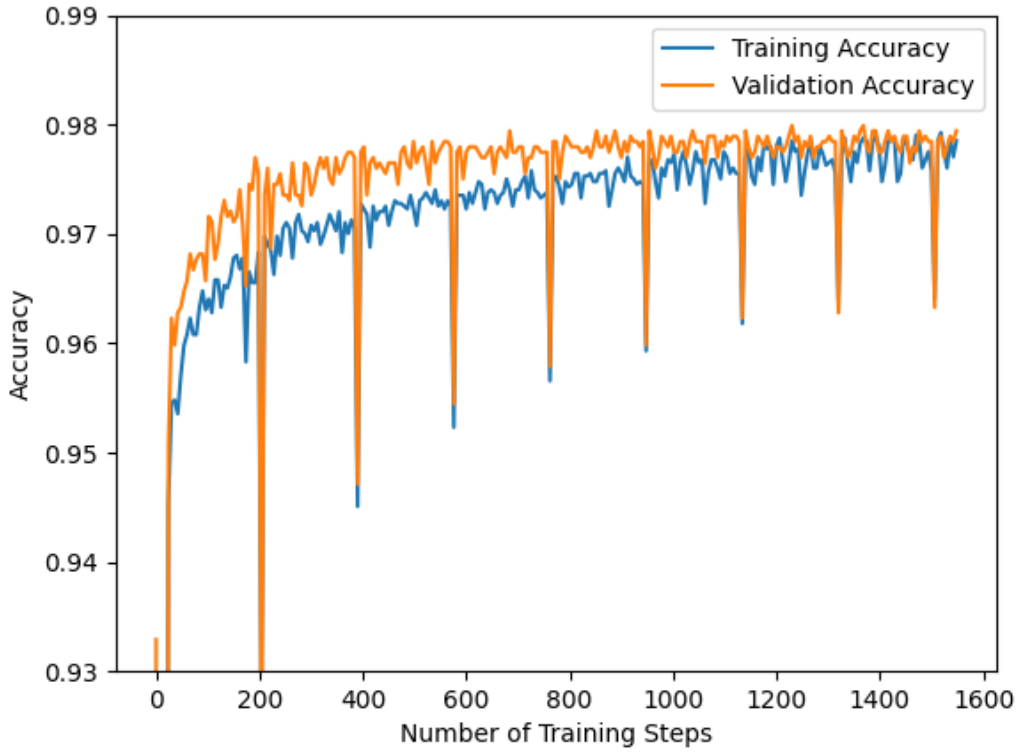


Figure 2: 2c) Training- and Validation Accuracy

2.3 Task 2d: Early stopping

The early stop kicks in at epoch 33.

2.4 Task 2e: Dataset shuffling

By comparing figure 2 and 3, it is clear that data shuffling reduces prediction noise. The implementation in the starter code used the same data set for every epoch, whereas the data shuffling addition ensured a unique dataset for every epoch. This introduces several improvements. Firstly, the network now trains on a much larger dataset, yielding better generalization, hence, better results on new data. Secondly, the number of wrong predictions will be higher on new data, hence, the gradient descent will be steeper compared with training on the same data. Lastly, the validation accuracy has less spikes. This is due to the fact that the probability of encountering a never seen sample is much lower. Also, the fact that the training set in reality is much larger, probably helps.

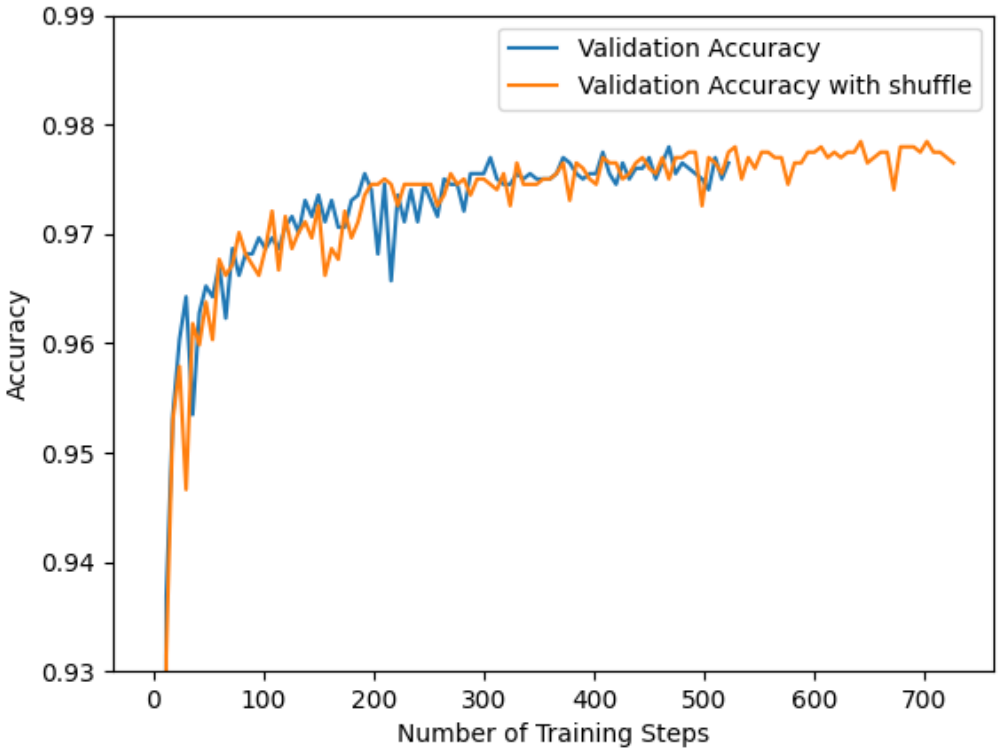


Figure 3: 2e) Training- and Validation Accuracy with Dataset Shuffling

3 Task 3: Softmax Regression through Gradient Descent

3.1 Task 3a and 3b: Softmax Regression

A plot of the training and validation loss is shown in figure 4.

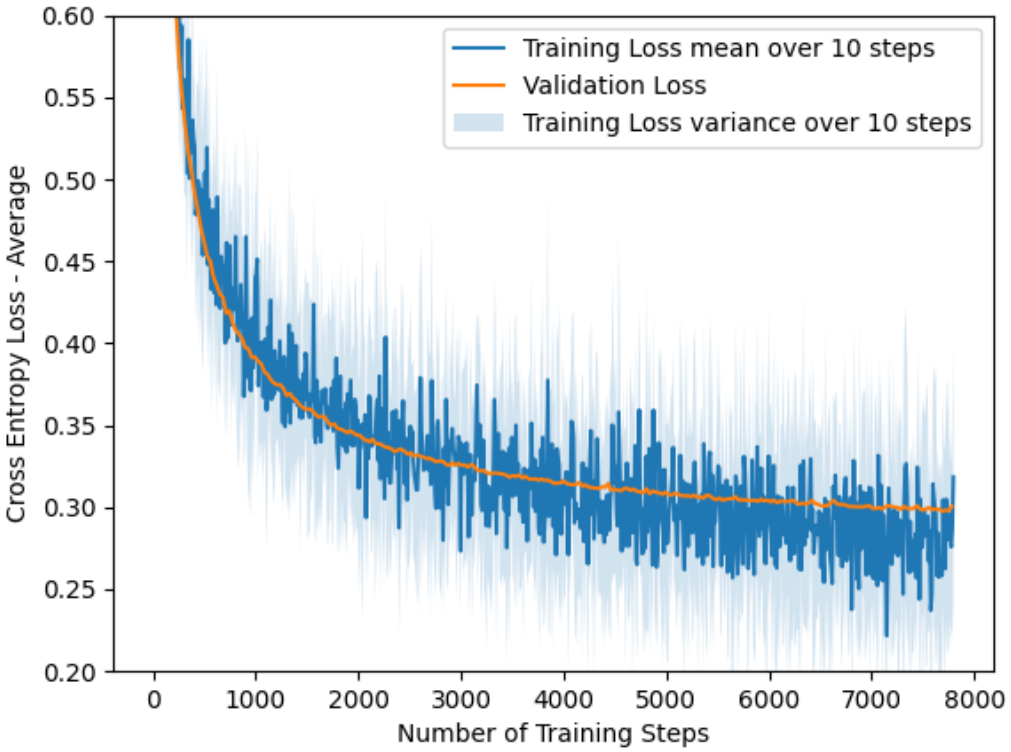


Figure 4: 3b) Training- and Validation Loss

3.2 Task 3c: Multi-class classification accuracy

Figure 5 shows the training and validation accuracy over training.

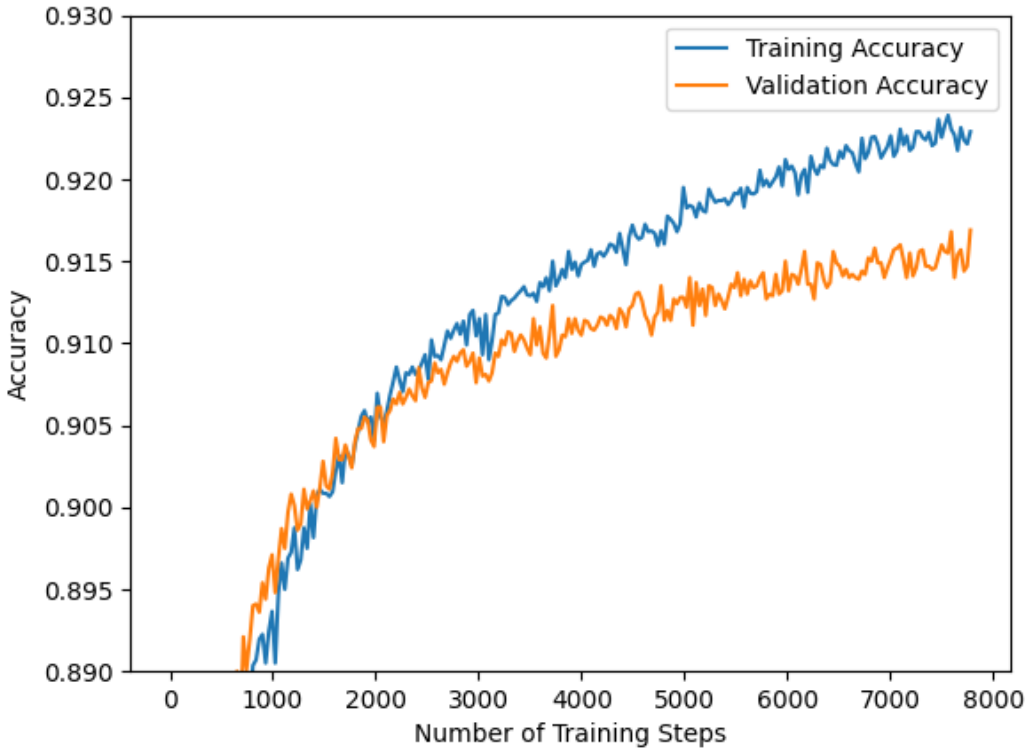


Figure 5: 3c) Training- and Validation Accuracy

3.3 Task 3d: Signs of overfitting

As seen by figure 5, the discrepancy between the validation and training accuracy increases over the number of training steps. This might be a sign that the network overfits on the given training data, and loses generalization. Hence, it tunes the weights to "perfectly" match the training data, and is very satisfied with itself. When faced with fresh validation data, reality hits it, and it turns out overfitting to match the training data perfectly was not such a good idea after all.

4 Task 4: regularization

4.1 Task 4a: Update term for Softmax Regression with L_2 regularization

Given the cost function

$$J(w) = C(w) + \lambda R(w), \tag{19}$$

where

$$R(w) = ||w||^2 = \sum_{ij} w_{ij}^2. \tag{20}$$

The regularizing gradient becomes

$$\frac{\partial J(w)}{\partial w} = \frac{\partial C(w)}{\partial w} + \lambda \frac{\partial R(w)}{\partial w}, \tag{21}$$

where

$$\frac{\partial R(w)}{\partial w} = 2w, \tag{22}$$

and $\frac{\partial C(w)}{\partial w}$ is given by equation 18.

Thus, the gradient becomes

$$\frac{\partial J(w)}{\partial w} = -x_j(y_k^n - \hat{y}_k^n) + 2\lambda w \tag{23}$$

4.2 Task 4b: L_2 regularization in backward pass

The weights obtained from using $\lambda = 0$ and $\lambda = 1$ are shown in figure 6 and 7, respectively. Clearly, the regularized weights are less noisy, and the digits are easier to interpret by human, visual inspection.

Intuitively, this is because regularization reduces overfitting, and hence, improves generalization. One way to think of this is that the regularized weights, to a larger extent, visualizes the networks "idea" of how the borders of a digit should look.

More specifically, regularization encourages the network to reduce the magnitude of the weights. This smoothes the image, making its visualization less noisy.

Figure 6: 4b) Weight Visualization without Regularization

Figure 7: 4b) Weight Visualization with Regularization

4.3 Task 4c: Model training with different values of λ

Figure 8 shows the validation accuracy for $\lambda = 1.0, 0.1, 0.01, 0.001$.

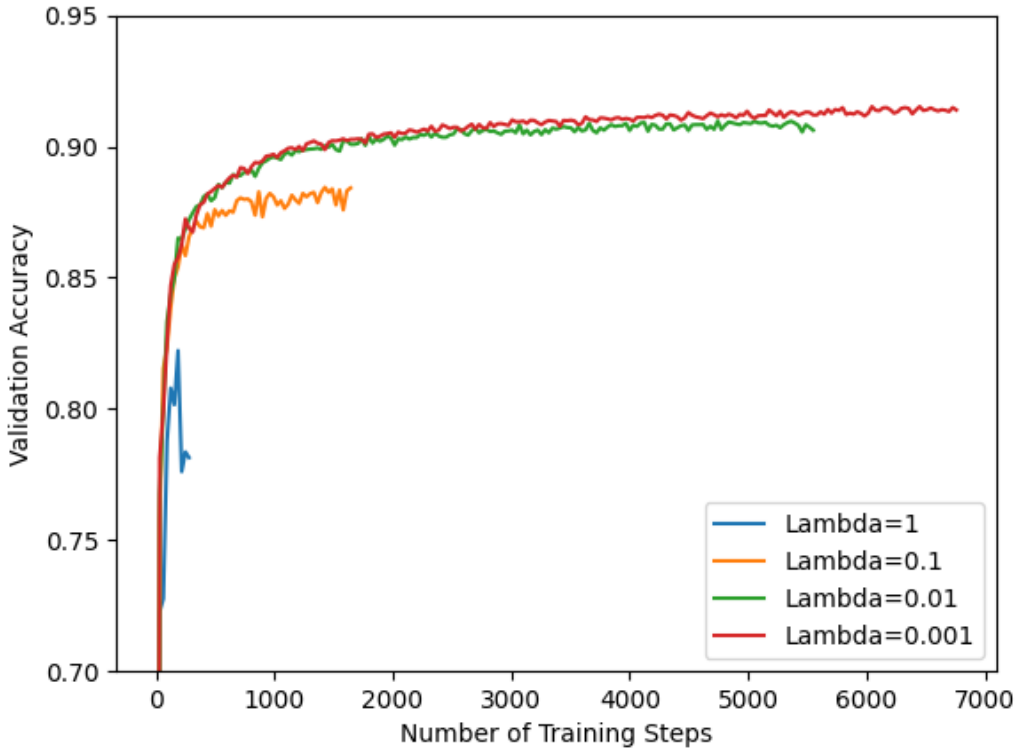


Figure 8: 4c) Validation Accuracy for different values of λ

4.4 Task 4d: Degraded validation accuracy for regularization

Figure 8 shows that the validation accuracy decreases for increasing values of λ. This happens because greater values of λ puts more emphasis on minimizing the weights, rather than the cost function. In other (possibly naive) words, it tells the gradient descent mechanism to care more about stepping in a direction that minimizes the weights, rather than a direction that minimizes the cost.

4.5 Task 4e: L_2-norm of weights

Figure 9 shows that the norm of the weight matrix decreases as λ increases. This makes sense, because greater values for λ penalizes weights with greater magnitudes more. Thus the magnitude of the weights, and accordingly the L_2 norm, will be smaller.

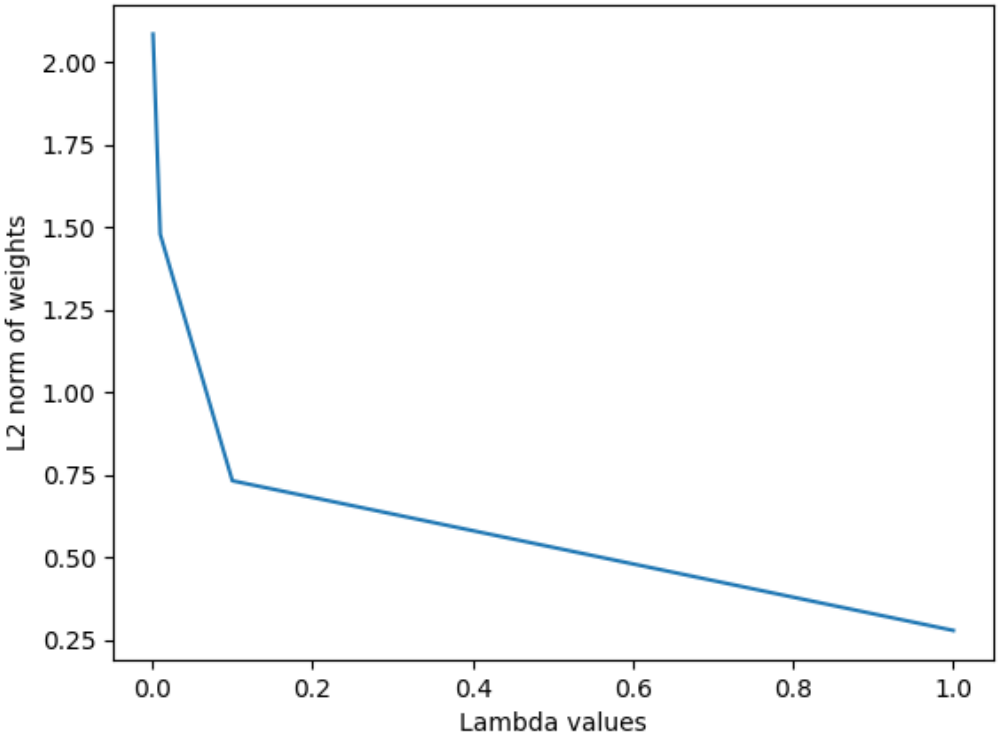


Figure 9: 4e) L_2-norms of weights for different values of λ