Assignment 01
TDT4265 - Datasyn og Dyplæring

Elias Mohammed Elfarri, Christopher Michael Vibe

Januar, 2021

Contents

1 Task 1

1.a

Given:

$$z_i = \sum_i^I w_i x_i \tag{1.1}$$

$$\hat{y}^n = f(z_i) = \frac{1}{1+e^{-z_i}} \tag{1.2}$$

$$C^n(w) = -(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y}^n)) \tag{1.3}$$

We use the chain rule and solve each partial derivative separately:

$$\frac{\partial C^n(w)}{\partial w_i} = \frac{\partial C^n}{\partial f}\frac{\partial f}{\partial z_i}\frac{\partial z_i}{\partial w_i}$$

First we find:

$$\frac{\partial C^n}{\partial f} = \frac{\partial(-(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y}^n)))}{\partial f}$$

$$= \frac{-y^n}{f} + \frac{1-y^n}{1-f} = \frac{-y^n(1-f)+f(1-y^n)}{f(1-f)}$$

$$= \frac{-y^n + y^n f + f - y^n f}{f(1-f)} = \underline{\underline{\frac{-y^n+f}{f(1-f)}}}$$

Secondly:

$$\frac{\partial f}{\partial z_i} = \frac{\partial(\frac{1}{1+e^{-z_i}})}{\partial z_i} = \frac{0*(1+e^{-z_i})+1*e^{-z_i}}{(1+e^{-z_i})^2}$$

$$= \frac{e^{-z_i}}{(1+e^{-z_i})^2} = (\frac{1}{1+e^{-z_i}})(\frac{e^{-z_i}}{1+e^{-z_i}}) = (\frac{1}{1+e^{-z_i}})(\frac{1-1+e^{-z_i}}{1+e^{-z_i}})$$

$$= (\frac{1}{1+e^{-z_i}})(\frac{1+e^{-z_i}}{1+e^{-z_i}} - \frac{1}{1+e^{-z_i}})$$

$$= (\frac{1}{1+e^{-z_i}})(1 - \frac{1}{1+e^{-z_i}})$$

$$= \underline{\underline{f(z_i)(1-f(z_i))}}$$

Thirdly:

$$\frac{\partial z_i}{\partial w_i} = \frac{\partial z_i}{\partial w_i} = \frac{\partial(\sum_i^I w_i x_i)}{\partial w_i} = \underline{\underline{x_i^n}}$$

Hence putting all these solutions together we get:

$$\frac{\partial C^n(w)}{\partial w_i} = \frac{-y^n+f}{f(1-f)} f(1-f)x_i^n = -(y^n - f)x_i^n = \underline{\underline{-(y^n-\hat{y}^n)x_i^n}}$$

1.b

Given score, softmax function and cost function:

$$z_k = \sum_i^I w_{k,i} x_i \tag{1.4}$$

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^K e^{z_{k'}}} \tag{1.5}$$

$$C^n(w) = -\sum_{k=1}^K y_k^n \ln(\hat{y}_k^n) \tag{1.6}$$

The chain rule used to find the gradient in this task was:

$$\frac{\partial C^n(w)}{\partial w_{k',j}} = \frac{\partial C^n}{\partial z_{k'}} \frac{\partial z_{k'}}{\partial w_{k',j}}$$

First we need to do the derivation for the softmax function for the different relevant cases:

if $k' = k$:

$$\frac{\partial \hat{y}_k}{\partial z_{k'}} = \frac{(e^{z_k})' \sum_{k'}^K e^{z_{k'}} - (\sum_{k'}^K e^{z_{k'}})' e^{z_k}}{(\sum_{k'}^K e^{z_{k'}})^2}$$

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

$$= (\frac{e^{z_k}}{\sum_{k'}^K e^{z_{k'}}})(\frac{\sum_{k'}^K e^{z_{k'}} - e^{z_{k'}}}{\sum_{k'}^K e^{z_{k'}}})$$

$$= \underline{\underline{\hat{y}_k^n (1 - \hat{y}_{k'}^n)}}$$

if $k' \neq k$:

$$\frac{\partial \hat{y}_k}{\partial z_{k'}} = \frac{0 * \sum_{k'}^K e^{z_{k'}} - e^{z_{k'}} e^{z_k}}{(\sum_{k'}^K e^{z_{k'}})^2}$$

$$= \underline{\underline{-\hat{y}_k^n \hat{y}_{k'}^n}}$$

Now we can use these scenarios in this partial derivation:

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

Expanding the sum based on $k = k'$ and $k \neq k'$:

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

$$= -y_k^n \frac{1}{\hat{y}_k^n} \frac{\partial \hat{y}_k^n}{\partial z_{k'=k}} - \sum_{k' \neq k}^{K} y_{k'}^n \frac{1}{\hat{y}_k^n} \frac{\partial \hat{y}_k^n}{\partial z_{k' \neq k}}$$

$$= -y_k^n \frac{1}{\hat{y}_k^n} \hat{y}_k^n (1 - \hat{y}_{k'}^n) - \sum_{k' \neq k}^{K} y_{k'}^n \frac{1}{\hat{y}_k^n} (-\hat{y}_k^n \hat{y}_{k'}^n)$$

$$= -y_k^n (1 - \hat{y}_{k'}^n) + \sum_{k' \neq k}^{K} y_{k'}^n \hat{y}_{k'}^n$$

$$= -y_k^n + \underbrace{y_k \hat{y}_{k'}^n + \sum_{k' \neq k}^{K} y_{k'}^n \hat{y}_{k'}^n}_{= \sum_{k'=k}^{K} y_k^n \hat{y}_{k'}^n}$$

$$= -y_k^n + \sum_{k'=k}^{K} y_k^n \hat{y}_{k'}^n = -y_k + \hat{y}_{k'=k}^n \underbrace{(\sum_{k}^{K} y_k^n)}_{=1}$$

$$= \underline{\underline{-y_k^n + \hat{y}_k^n}}$$

And in the end we need to find the last partial derivative both when $k' = k$ and when it is $k' \neq k$, which gives the same result:

$$\frac{\partial z_{k'}}{\partial w_{k',j}} = \frac{\partial(\sum_j^J w_{k',j} x_j^n)}{\partial w_{k',j}} = \underline{\underline{x_j^n}}$$

Hence the final solution is:

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

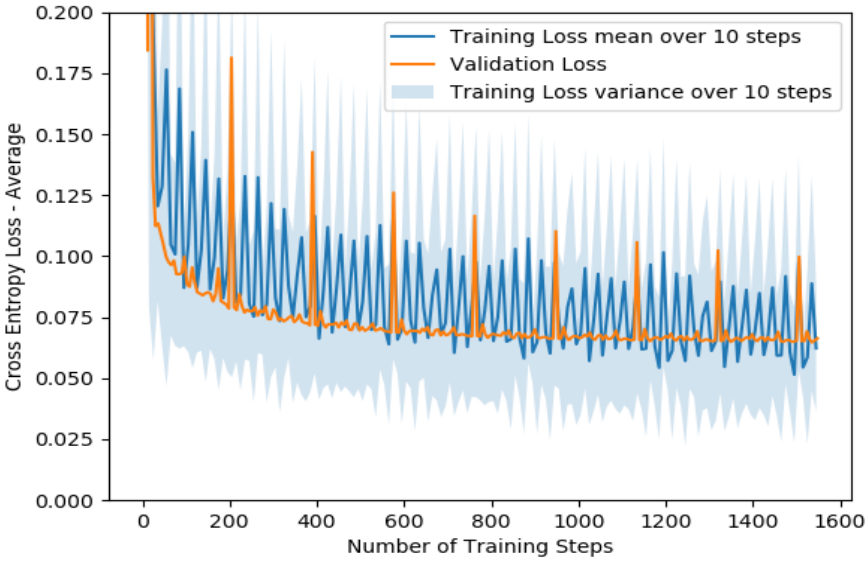
Another valid approach is to expand the natural logarithm at the start and deal with those two terms separately.

2 Task 2

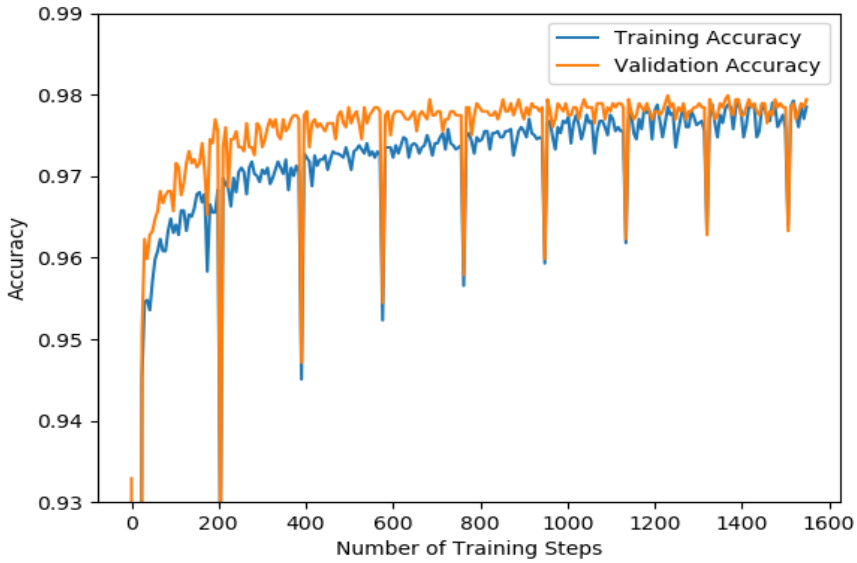
2.a

Just code

2.b


Training Loss mean over 10 steps
Validation Loss
Training Loss variance over 10 steps
Cross Entropy Loss - Average
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
200
400
600
800
1000
1200
1400
1600
Number of Training Steps

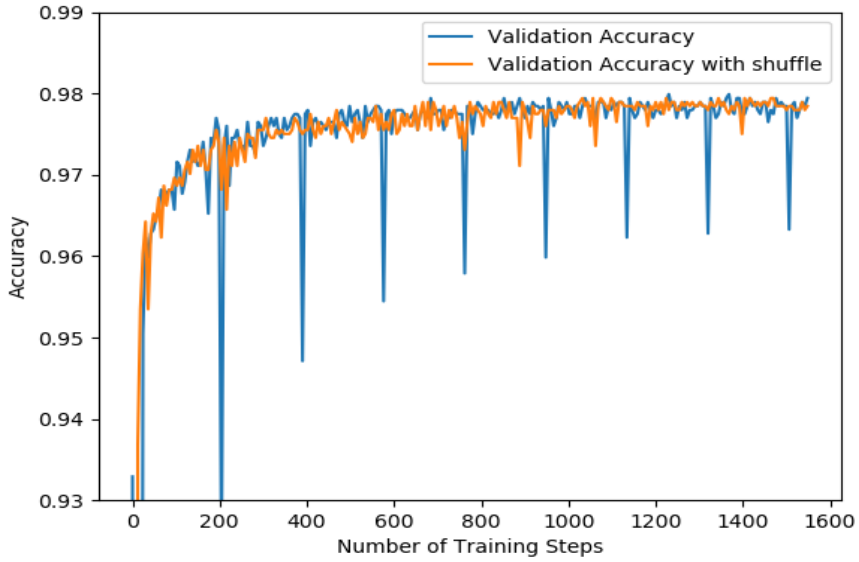
2.c


Training Accuracy
Validation Accuracy
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0
200
400
600
800
1000
1200
1400
1600
Number of Training Steps

2.d

After 33 epochs early stopping kicks in with train accuracy of 97.5% and validation accuracy of 97.9%. Without early stopping the validation accuracy goes down if it continues training to 500 epochs. This makes sense as the model is being over-fitted.

2.e

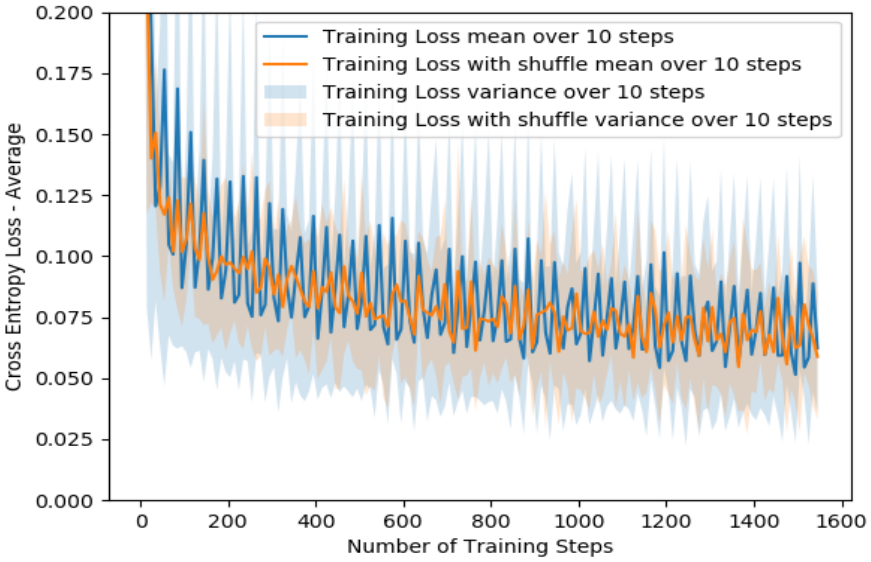


From the graph there can be observed two patterns:

The first pattern is observed in both curves in the form of static noise. The noise is relatively low in variance, and is exemplified in training steps 50 to 100 in a relatively smooth oscillation. This is most likely due to gradient descent overshoot, perhaps from an aggressive learning rate. Another possible reason could be that the batch sizes are too small, and should be further averaged. This is however, expected in as part of any training process.

The second pattern is marked by the validation accuracy without the shuffling scheme. The accuracy drops around every 200 training steps, and mirrors the cost giving a great surge at these points. This effect could be coming from harder batches that occur periodically during the testing phase.

More formally this is most likely due to the existence of multiple local optima, where the current local minima doesn't generalize as well as the local



minima from the last epoch. The gradient descent periodically seeks out a local optimum, and with time, manages to break out in search for a better local optimum. In a marble analogy, where a marble is rolling on a landscape of solution space, it is going from one local point of low potential to the next. By shuffling the data the gradient shifts help push this marble out of the local optimum, with improved direction because the landscape itself has changed on each epoch. In effect the gradients guide the marble on a better and more averaged out path that is less erratic.

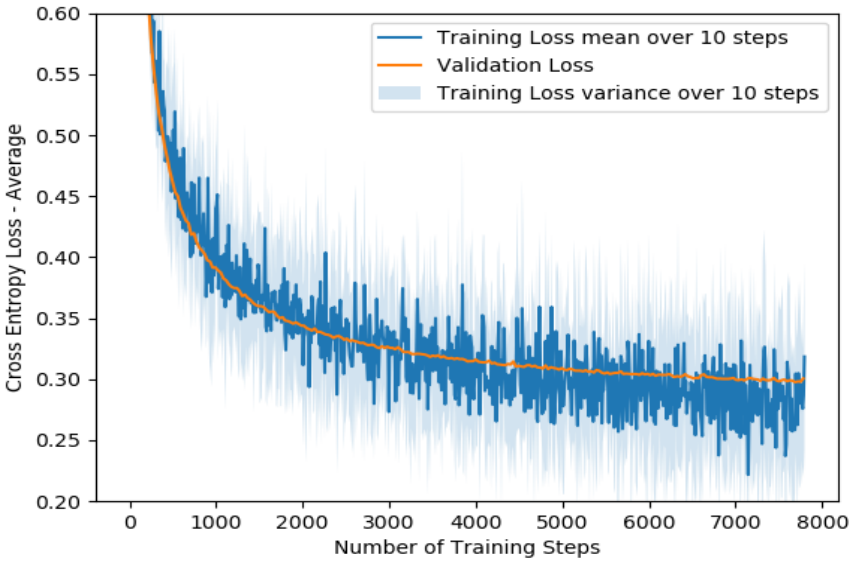
An additional point is that the input sequence will be decoupled from the learning process when shuffling. For example memorizing the answer to n sequential questions as opposed to answering each question independent of sequence.

3 Task 3

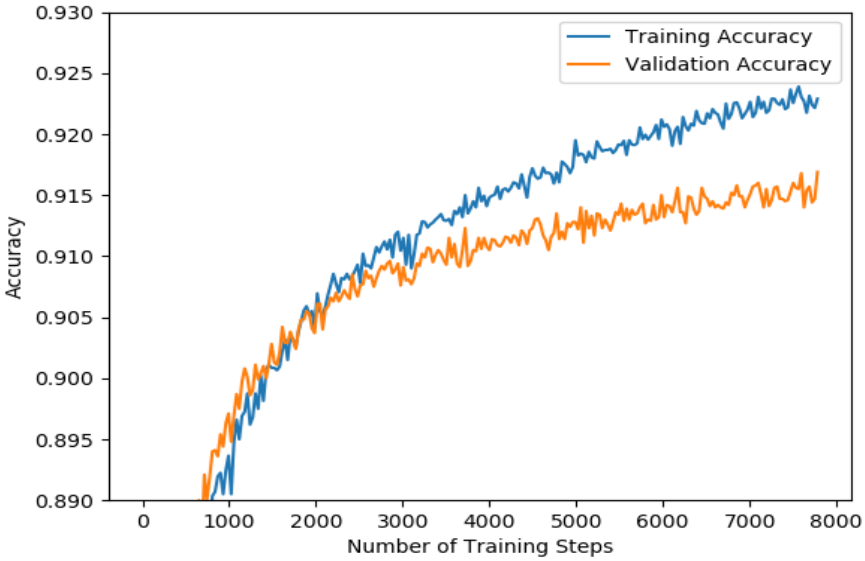
3.a

Coding assignment.

3.b



3.c


Training Accuracy
Validation Accuracy
Accuracy
Number of Training Steps
0.930
0.925
0.920
0.915
0.910
0.905
0.900
0.895
0.890
0
1000
2000
3000
4000
5000
6000
7000
8000

3.d

The main sign of over-fitting in the accuracy graph from 3c is that there is a gap that is forming between the training accuracy and the validation accuracy. This does not mean that the model is over-fitted, because the validation accuracy may be flattening out. It does however indicate that there is potential for such development if the number of epochs is increased for training.

4 Task 4

4.a

We only need to figure out the regularization derivative as the gradient for the soft-max loss function is found in 1b:

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

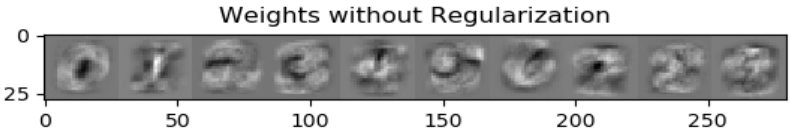
$$\frac{\partial C(w)}{\partial w_{k,j}} = -(y_k^n - \hat{y}_k^n)x_j^n$$

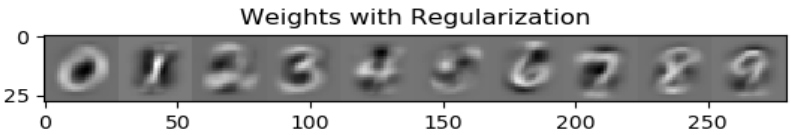
$$\frac{\partial R(w)}{\partial w_{k,j}} = \frac{\partial(\sum_{k,j} w_{k,j}^2)}{\partial w_{k,j}} = 2w_{k,j}$$

$$\frac{\partial J(w)}{\partial w_{k,j}} = \underline{\underline{-(y_k^n - \hat{y}_k^n)x_j^n + 2\lambda w_{k,j}}}$$

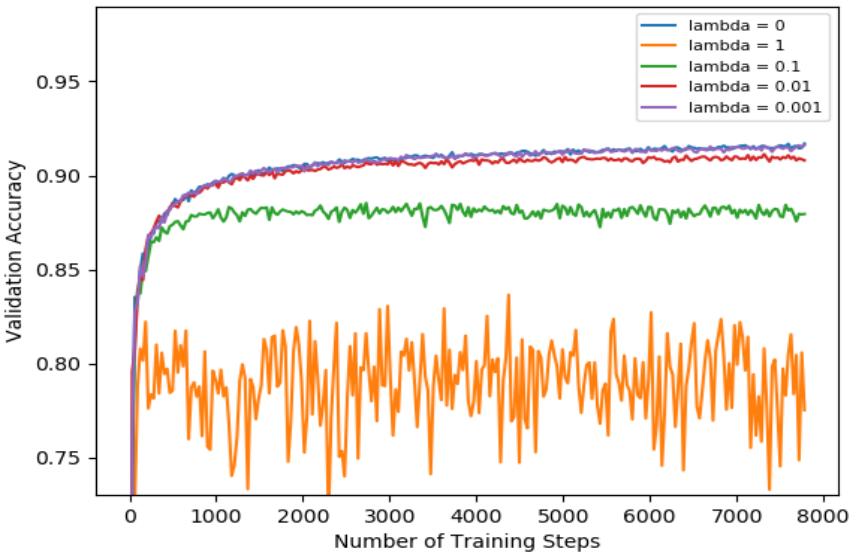
4.b

From the two graphs below the model weights are presented in an intuitive manner, both with and without L2 regularization. Note that the first figure, appears more smudged and pixelated than the figure with regularization. This is a sign that the model has been more generalized. The regularization term introduces a penalty to the cost function such that the optimization problem is restricted. This way it motivates the cost function to find a more generalized solution in a slightly modified solution space. Mathematically this is because a sum of squares is always less if the values are similar in magnitude, forcing the model to rely on each weight more or less equally. If the gradient update only contained the regularization part, all weights would tend towards zero.


Weights without Regularization
0
25
0
50
100
150
200
250


Weights with Regularization
0
25
0
50
100
150
200
250

4.c

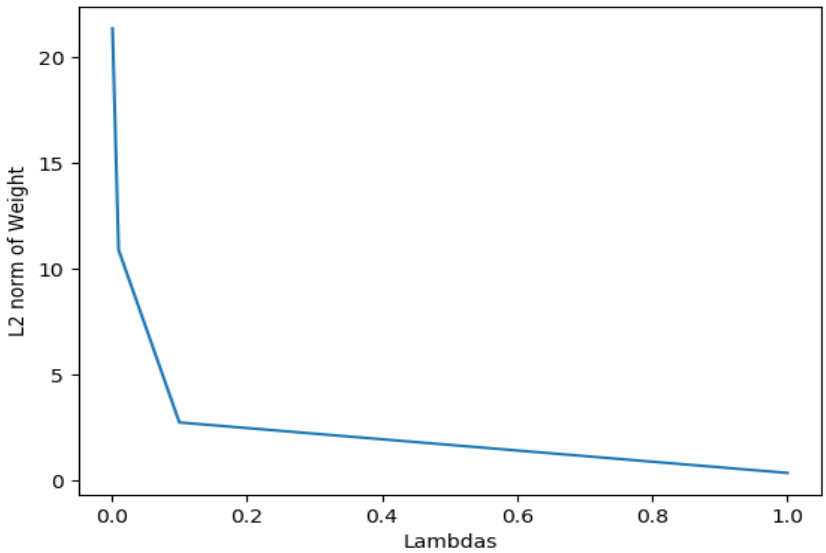

lambda = 0
lambda = 1
lambda = 0.1
lambda = 0.01
lambda = 0.001
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0
1000
2000
3000
4000
5000
6000
7000
8000
Number of Training Steps

4.d

The lambda value works as a percentage term, in other words, how much of an influence the penalty term will have on the cost function. This way a high lambda will limit the complexity of the model, and hence may risk under-fitting the validation set. From our figure, this phenomenon can be observed for $\lambda = 1$. The penalty is simply too high and the model was not able to generate a complex enough hypothesis during the training to make useful predictions when exposed to non-training data. As lambda is decreased, so does the validation accuracy increase. This indicates that regularization can be damaging to the model, and the lambda parameter needs to be calibrated with the intention of improving validation accuracy/lowering validation loss.

Above we have established that the lambda value acts as complexity control for the models cost function, but any amount of L2 regularization seems to be damaging to the model. This could indicate that the model is too simple, and is not over-fitting. In other words, the model is already limited enough by its complexity as it is, and does not need to be simplified to achieve generalization. The model could be made more complex in many ways, like introducing another intermediary layer. If complexity is increased then, it may be that L2 regularization becomes useful. This explanation is backed up by the fact that both the training and validation accuracy's are almost the same at 92.3% and 91.6% respectively. (Numbers from sub-task 3c)

4.e



L2 regularization is also called weight-decay, because for each gradient update, the weights are reduced an amount towards zero relative to the rate of lambda. This means as the lambda gets bigger, then the weight-decay becomes even faster. This tendency is clearly illustrated in the graph below. We can observe that as the lambda parameter is increased the L2 norm of the weight has an exponential decay relationship with lambda.