Computer Vision and Deep Learning

TDT4265
Assignment 1
MNIST Classification

Halvor Bakken Smedås

January 30, 2022

Task 1

Task 1a)

`Derive the gradient for Logistic Regression. To minimize the cost function with gradient descent, we require the gradient of the cost function. Show that for Equation 3 (Binary Cross Entropy Loss), the gradient is:`

$$\frac{\partial C^n(w)}{\partial w_i} = -\left(y^n - \hat{y}^n\right) x_i^n$$

when the output of our network is given by

$$\hat{y} = f(x) = \frac{1}{1 + e^{-w^T x}},\ w^T x = \sum_i^I w_i \cdot x_i$$

Equation 3 - The Binary Cross Entropy loss is defined as:

$$C(w) = \frac{1}{N} \sum_{n=1}^{N} C^n(w), \text{ where } C^n(w) = -\left(y^n \ln\left(\hat{y}^n\right) + \left(1 - y^n\right) \ln\left(1 - \hat{y}^n\right)\right)$$

Through the chain rule, we have that

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

We utilise the given hint $\frac{\partial f(x^n)}{\partial w_i} = x_i^n f(x^n)\left(1 - f(x^n)\right)$, and identify that $f(x^n) \equiv \hat{y}^n$, such that our expression can be rewritten as

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

Now, we only need to unravel the factor $\frac{\partial C^n(w)}{\partial \hat{y}^n}$, and we'll be at the full gradient expression:

$$\frac{\partial C^n(w)}{\partial \hat{y}^n}, \text{ where } C^n(w) = -(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y}^n))$$

We start of by reorganising $C^n(w)$ to allow for easy derivation of individual terms:

$$\begin{aligned} C^n(w) &= -(y^n \ln(\hat{y}^n) + (1-y^n)\ln(1-\hat{y}^n)) \\ &= -(y^n \ln(\hat{y}^n) + (\ln(1-\hat{y}^n) - y^n \ln(1-\hat{y}^n))) \\ &= -(y^n \ln(\hat{y}^n) + \ln(1-\hat{y}^n) - y^n \ln(1-\hat{y}^n)) \\ &= -y^n \ln(\hat{y}^n) - \ln(1-\hat{y}^n) + y^n \ln(1-\hat{y}^n) \end{aligned}$$

Now we can rewrite the partial derivative $\frac{\partial C^n(w)}{\partial \hat{y}^n}$ as

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

We start by taking out the *'constants'* (y^n and term signs), before we proceed to evaluate each individual derivative:

$$\begin{aligned} &= -y^n \frac{\partial}{\partial \hat{y}^n}(\ln \hat{y}^n) && -1\frac{\partial}{\partial \hat{y}^n}(\ln(1-\hat{y}^n)) && +y^n \frac{\partial}{\partial \hat{y}^n}(\ln(1-\hat{y}^n)) \\ &= -y^n \frac{1}{\hat{y}^n} && -1\frac{1}{1-\hat{y}^n}\frac{\partial}{\partial \hat{y}^n}(1-\hat{y}^n) && +y^n \frac{1}{1-\hat{y}^n}\frac{\partial}{\partial \hat{y}^n}(1-\hat{y}^n) \\ &= -y^n \frac{1}{\hat{y}^n} && -1\frac{1}{1-\hat{y}^n}(-1) && +y^n \frac{1}{1-\hat{y}^n}(-1) \\ &= -\frac{y^n}{\hat{y}^n} && +\frac{1}{1-\hat{y}^n} && -\frac{y^n}{1-\hat{y}^n} \end{aligned}$$

In other words, we have that

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

We can now combine the two partial derivative expressions we've computed; $\frac{\partial C^n(w)}{\partial \hat{y}^n}$ and $\frac{\partial \hat{y}^n}{\partial w_i}$:

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

Task 1b)

`Derive the gradient for Softmax Regression. For the multi-class cross entropy cost in Equation 5 (Categorical Cross Entropy Loss), show that the gradient is:`

$$
\frac{\partial C^n(w)}{\partial w_{kj}} = -x_j^n\left(y_k^n - \hat{y}_k^n\right)
$$

Equation 5 - The Categorical Cross Entropy Loss function is given by

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

Additionally, we have that $\hat{y}$ is a vector, with elements $\{\hat{y}_k\}_{k=1}^K$ Each of which can be represented by

$$
\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}}, \text{ where } z_k = w_k^T \cdot x = \sum_i^I w_{ki} \cdot x_i
$$

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

Let's begin by declaring the chain we have to get through...

$$\frac{\partial C^n(w)}{\partial w_{ij}} = \frac{\partial C^n(w)}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} \tag{2}$$

That is, the derivative of the cost (cross entropy loss) wrt. the network output vector element z_i times the derivative of the output vector element wrt. to the corresponding weights. From here, let's begin with the first of the two. By the law of derivative summation, we have:

$$\frac{\partial C^n(w)}{\partial z_i} = -\sum_{k=1}^{K} \left[y_k^n \frac{\partial \ln(\hat{y}_k^n)}{\partial z_i} \right] \tag{3}$$

We utilize the logarithmic derivative ($\frac{\partial \ln a}{\partial b} = \frac{1}{a}\frac{\partial a}{\partial b}$), and get

$$\frac{\partial C^n(w)}{\partial z_i} = -\sum_{k=1}^{K} \left[\frac{y_k^n}{\hat{y}_k^n} \frac{\partial \hat{y}_k^n}{\partial z_i} \right] \tag{4}$$

From here, let's focus on the inner partial derivative — the derivative of the softmax function wrt. the output vector element — and leave the rest unchanged...

$$\frac{\partial \hat{y}_k^n}{\partial z_i} = \frac{\partial}{\partial z_i} \frac{e^{z_k}}{\sum_{k'=1}^{K} [e^{z_{k'}}]} = \frac{\partial}{\partial z_i} \frac{e^{z_k}}{[e^{z_1} + \ldots + e^{z_i} + \ldots + e^{z_K}]} \text{ For clarity, expand the summation} \tag{5}$$

Before we carry on from this, we need to consider how this all behaves in relation to i and k. i and k in the context of z illustrates which element of the z-vector we are working with (technically, the z^n-vector as this expression also spans $n \in N$, but for the sake of this computation this is merely a technicality). The point here, is that i and k may overlap, as they're indexing the same vector, and how we differentiate this expression boils down to evaluating the two cases of $k \neq i$ and $i = k$.

$$\frac{\partial \hat{y}_k^n}{\partial z_i} = \begin{cases} \frac{0[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]-[0+\ldots+e^{z_i}+\ldots+0]e^{z_k}}{[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]^2} & \{k \neq i\} \\ \frac{e^{z_i}[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]-[0+\ldots+e^{z_i}+\ldots+0]e^{z_k}}{[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]^2} & \{k = i\} \end{cases} \tag{6}$$

$$= \begin{cases} \frac{0-e^{z_i}e^{z_k}}{[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]^2} & \{k \neq i\} \\ \frac{e^{z_i}[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]-(e^{z_i})^2}{[e^{z_1}+\ldots+e^{z_i}+\ldots+e^{z_K}]^2} & \{k = i\} \end{cases} \tag{7}$$

Now that we've handled the derivatives of the two scenarios and handled the variable dependencies, we can again revert to the more concise summation form:

$$\frac{\partial \hat{y}_k^n}{\partial z_i} = \begin{cases} \frac{-e^{z_i}e^{z_k}}{\sum_{k'=1}^{K}\left[e^{z_{k'}}\right]^2} & \{k \neq i\} \\ \frac{e^{z_i}\sum_{k'=1}^{K}\left[e^{z_{k'}}\right]-(e^{z_i})^2}{\sum_{k'=1}^{K}\left[e^{z_{k'}}\right]^2} & \{k = i\} \end{cases} \tag{8}$$

We're not quite done with it yet though, as we can go on and simplify even further, by recognising the hidden factors within these expressions:

$$\frac{\partial \hat{y}_k^n}{\partial z_i} = \begin{cases} \frac{-e^{z_i} e^{z_k}}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]^2} = -\frac{e^{z_i}}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]} \frac{e^{z_k}}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]} = -\hat{y}_i^n \hat{y}_k^n & \{k \neq i\} \\ \frac{e^{z_i} \sum_{k'=1}^{K} \left[e^{z_{k'}}\right] - \left(e^{z_i}\right)^2}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]^2} = \frac{e^{z_i} \left(\sum_{k'=1}^{K} \left[e^{z_{k'}}\right] - e^{z_i}\right)}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]^2} & \\ = \frac{e^{z_i}}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]} \left(\frac{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]} - \frac{e^{z_i}}{\sum_{k'=1}^{K} \left[e^{z_{k'}}\right]} \right) = \hat{y}_i^n \left(1 - \hat{y}_i^n\right) & \{k = i\} \end{cases} \tag{9}$$

With that out of the way, lets get back to the original first part of the gradient chain:

$$\frac{\partial C^n(w)}{\partial z_i} = -\sum_{k=1}^{K} \left[\frac{y_k^n}{\hat{y}_k^n} \frac{\partial \hat{y}_k^n}{\partial z_i} \right] \tag{10}$$

We can now easily compute the full summation by excluding the term where $\{i = k\}$ and adding that separately:

$$\frac{\partial C^n(w)}{\partial z_i} = -\sum_{\substack{k=1 \\ k \neq i}}^{K} \left[\frac{y_k^n}{\hat{y}_k^n} \left(-\hat{y}_i^n \hat{y}_k^n\right) \right] + \frac{y_i^n}{\hat{y}_i^n} \hat{y}_i^n \left(1 - \hat{y}_i^n\right) \tag{11}$$

$$= -\sum_{\substack{k=1 \\ k \neq i}}^{K} \left[-y_k^n \hat{y}_i^n \right] + y_i^n \left(1 - \hat{y}_i^n\right) \tag{12}$$

$$= \sum_{\substack{k=1 \\ k \neq i}}^{K} \left[y_k^n \hat{y}_i^n \right] - y_i^n \left(1 - \hat{y}_i^n\right) \tag{13}$$

$$= \sum_{\substack{k=1 \\ k \neq i}}^{K} \left[y_k^n \hat{y}_i^n \right] - y_i^n + y_i^n \hat{y}_i^n \tag{14}$$

$$= \sum_{k=1}^{K} \left[y_k^n \hat{y}_i^n \right] - y_i^n \tag{15}$$

$$= \hat{y}_i^n \sum_{k=1}^{K} \left[y_k^n \right] - y_i^n \tag{16}$$

$$\tag{17}$$

The special case of $\sum_{k=1}^{K} \left[y_k^n \right]$ is the known sum of the softmax function on K classes, namely the sum of probabilities, which of course is equal to 1

$$\frac{\partial C^n(w)}{\partial z_i} = \hat{y}_i^n \sum_{k=1}^{K} [y_k^n] - y_i^n \tag{18}$$

$$\frac{\partial C^n(w)}{\partial z_i} = \hat{y}_i^n - y_i^n \tag{19}$$

$$\tag{20}$$

And with that, we now only need the trivial

$$\frac{\partial z_i}{\partial w_{ij}} = x_j \tag{21}$$

And then we have the full chain:

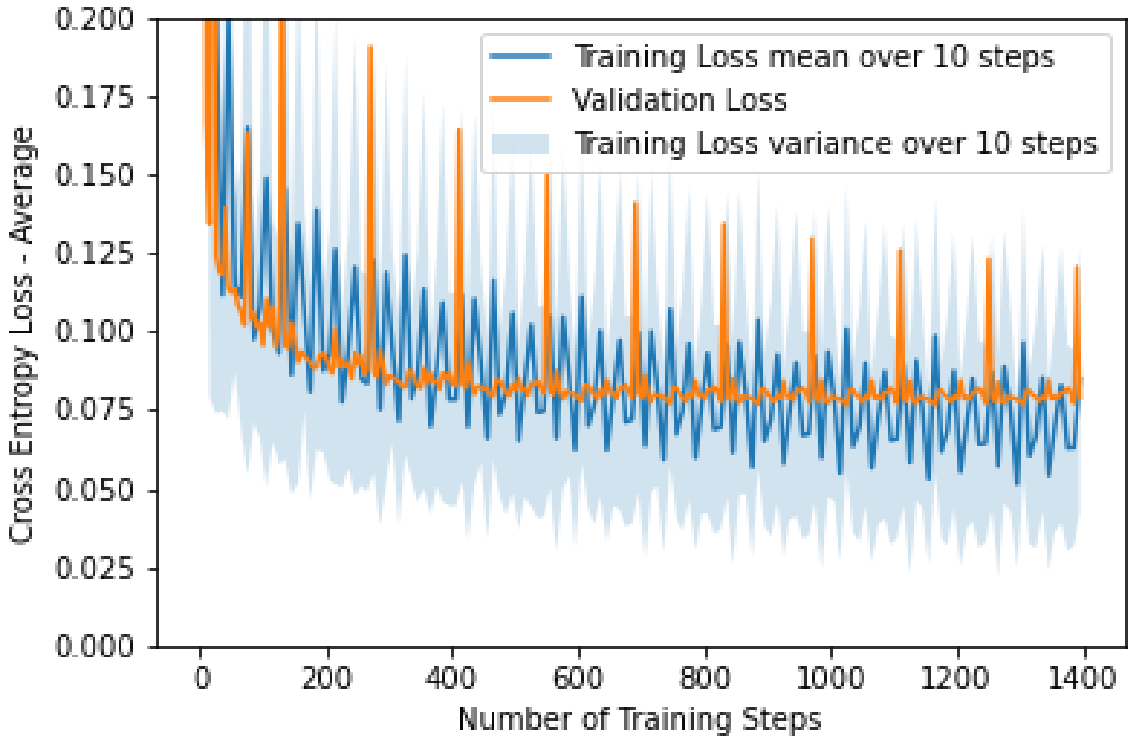
$$\frac{\partial C^n(w)}{\partial w_{ij}} = \frac{\partial C^n(w)}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} = (\hat{y}_i^n - y_i^n)\, x_j \tag{22}$$

which, if we reorder around a little grants us with

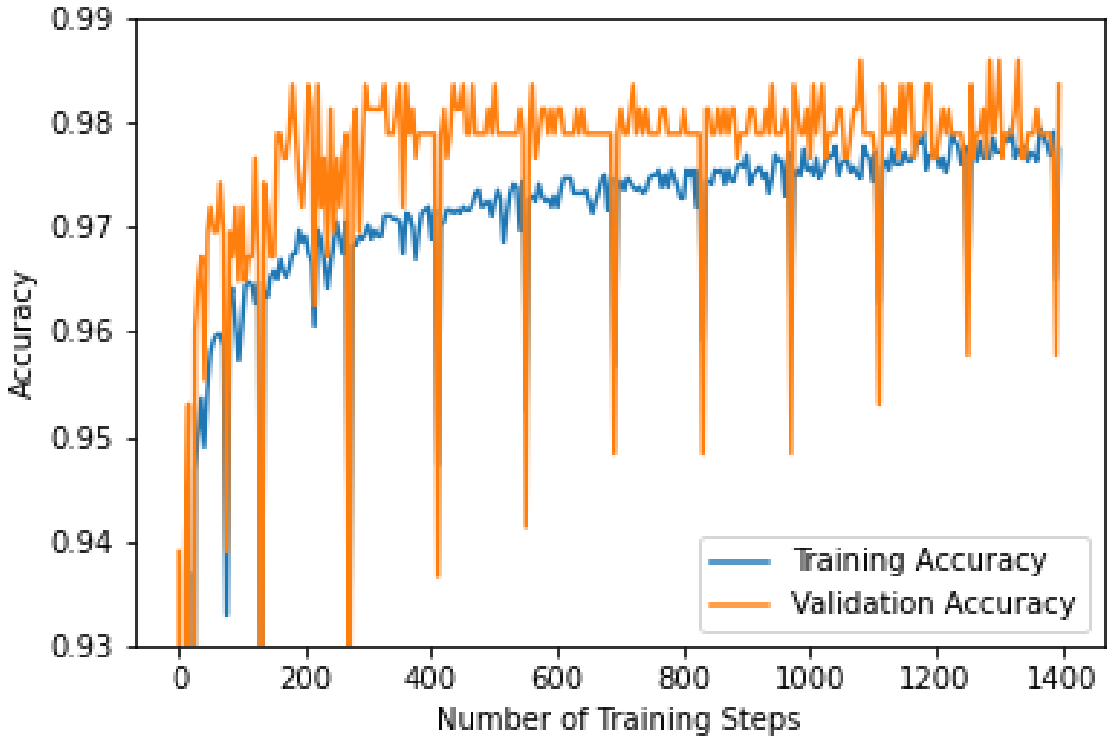
$$\frac{\partial C^n(w)}{\partial w_{ij}} = -x_j\, (y_i^n - \hat{y}_i^n) \tag{23}$$

Task 2

Task 2b)



Task 2c)



Task 2d)

With early stopping enabled, training stops at step 75.

With the batch shuffling enabled too, training seems to stop arbitrarily in the step range [65-195]

Observing again how the train loss and validation loss progress, and approximately where they start showing indications of overfitting - which we would put somewhere around step 600, we consider the early stopping criteria a little bit too sensitive. This is further indicated by the improved validation accuracy after the early stopping range when early stopping is disabled.

Task 2e)

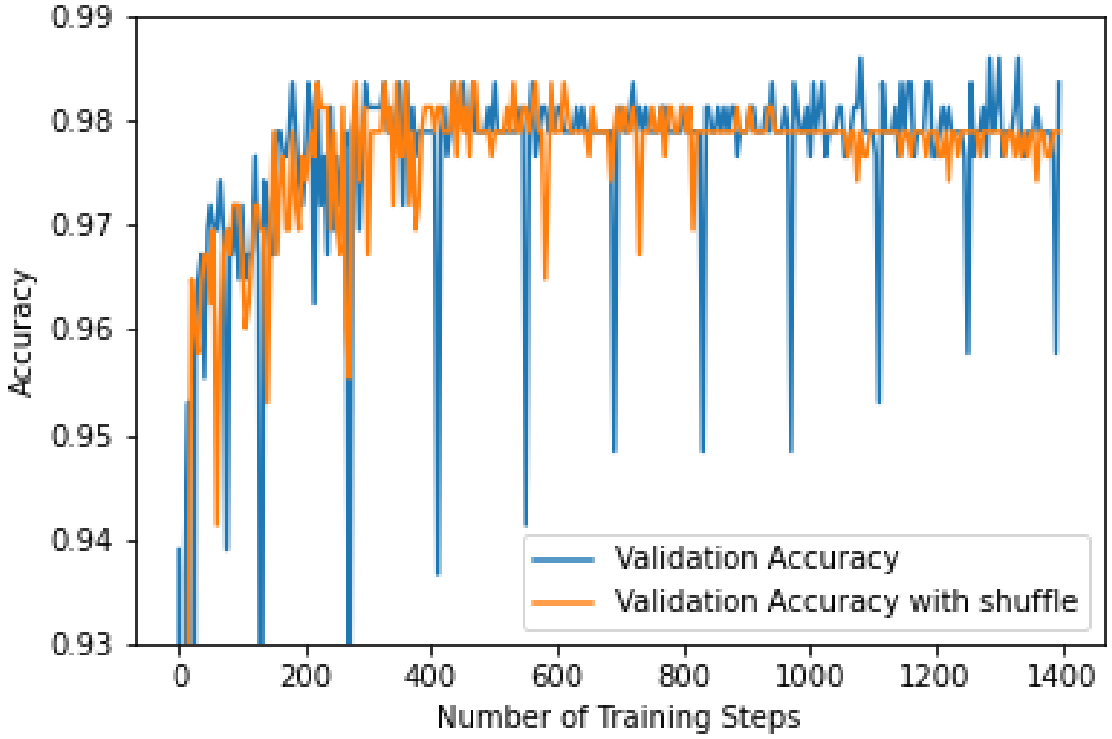
The reduction of "spikes" in accuracy progression we see when dataset shuffling is enabled can be a result of several factors.

My main hypothesis is that the gradient descent "drifts" of from the *ideal* path every epoch. The reason simply being that the individual mini-batches don't represent the various extremities of the dataset.

The reason we're only seeing 10 spikes, is because we only do validation steps every 5th step through training, while there are 28 steps per epoch, which means we only track the epoch start every 5th epoch ($x28 \bmod 5 = 0 \implies x = 5$), while in fact during training there's one spike for every epoch.

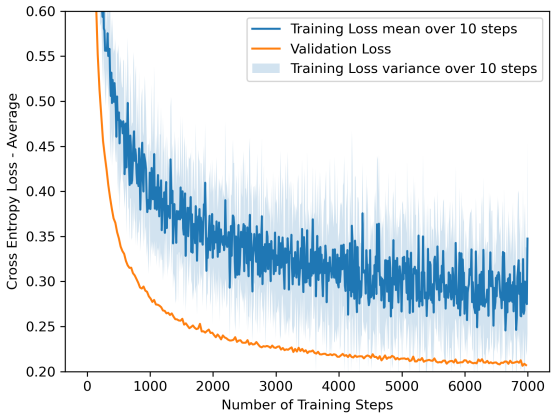
To me this is a clear indication that my hypothesis is strong. The samples at the start of every epoch is partially "unlearnt" as the latter samples of the epoch are biased towards different niches.

Using dataset shuffling, we circumvent this issue by continually — by the balance of probability — keep the mini-batches as representative of the whole dataset as possible.

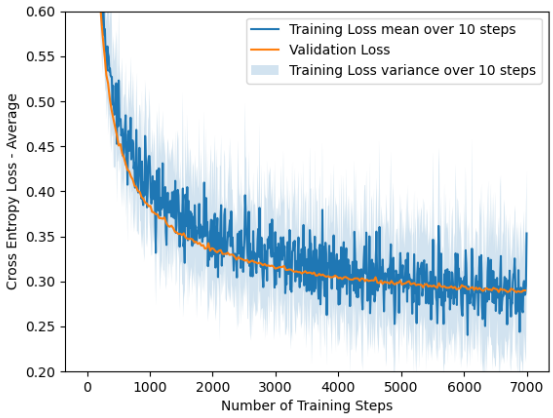


Task 3

Task 3b)

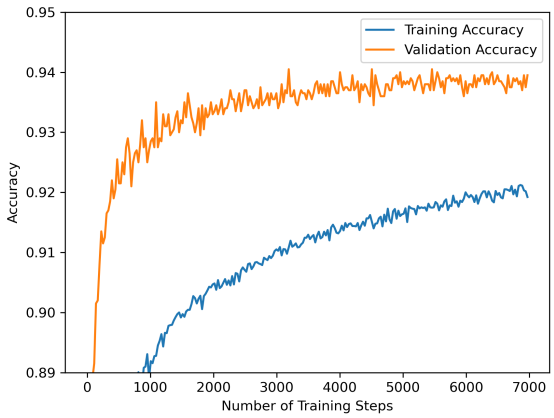


(a) Validation set sequentially selected, last 2000

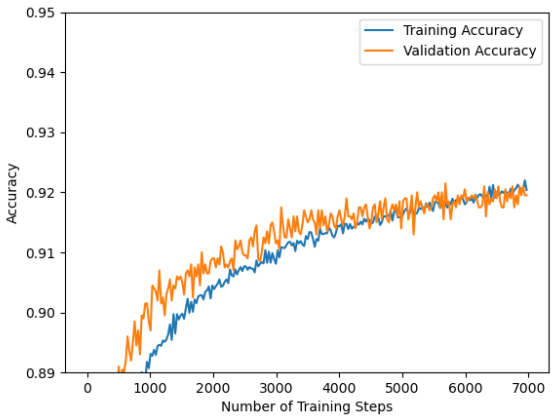


(b) Validation set stochastically sampled

Task 3c)



(a) Validation set sequentially selected, last 2000



(b) Validation set stochastically sampled

Task 3d)

No. Initially, there's no sign of overfitting. A typical indication of overfitting is that the training accuracy first of all surpasses the validation accuracy — and second of all — validation accuracy stagnates wile training accuracy continues to improve. Another very typical sign of overfitting is that the validation loss deviates from the training loss — where the mean validation loss slows down it's decrease compared to the training loss.

In fact we find that the validation accuracy is surprisingly high compared to the training accuracy. The same also applies for the respective losses. Generally speaking, having a validation accuracy be higher than the training accuracy isn't unheard of, and can be seen as a win for us as the trainers

of the model. However, with this significant a leap in accuracy from training to validation, it is highly likely there is a bug in the evaluation code — Yes, we apply a very healthy dose of pessimism when training, and this scenario simply is too good to be true.

In this scenario, we haven't been careful enough with the data we use in evaluation, making it seem as if we're performing very very well on unseen samples. However, this isn't really the case...

In particular, we use a predefined set of samples for evaluation — Doing so can be perfectly fine, and is often how we approach test and validation sets. However, we are only using a subset of the full set as our validation set. As soon as we make a sub-selection, there are two things we need to make sure of. The point of any validation set is to have a selection of samples from the problem domain (here being handwritten digits) that are distinct from the training set. Hence, the most important thing with such a set, is that it's: `1.` distinct from the training set, and `2.` that it manifests the full domain.

With our code, only the first of the two clauses are ensured. `2.` is not upheld as we're only sampling 2000 samples from the end of the original mnist test set. As we were not the ones assembling the test set, we cannot be sure the samples are fairly spanning the domain to its fullest. For all we know, the 2000 last samples of the test set represents digits only written by one person, biasing the validation to said person's vision of what "true digits" look like. Or even worse, we might think we sample 2000 samples uniformly spanning the 10 different classes of the mnist dataset, while in fact we only get 1000 zeros and 1000 ones.

The latter of those two cases, we tested in code, by summing the one-hot encoded labels of the validation set:

	0	1	2	3	4	5	6	7	8	9
%	10.35	11.50	09.90	10.35	09.70	08.45	10.10	10.75	09.35	09.55

This seems within reason in terms of class stability in the validation set. But we have no way of ensuring that the individual samples themselves don't tend to some niche of the *hand written digit domain*.

Hence, we deemed it worth the while to implement a stochastic sampling variation of dataset loading. Where we still sample only 2000 samples, but they are selected at random without replacement across the 10000 available samples in the original test set. Doing so should in theory aid in representing the full span of the test set, while it also, probabilistically keeps the distribution of classes similar to that of the whole test set too. Above, we've included the plots with and without stochastic validation set selection. Where we in the stochastic one see a slight tendency towards overfitting

Task 4

Task 4a)

$$\frac{\partial J(w)}{\partial w} = \frac{\partial C(w)}{\partial w} + \lambda \frac{\partial R(w)}{\partial w} \tag{24}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial C^n(w)}{\partial w} \right] + \lambda \frac{\partial \|w\|^2}{\partial w} \tag{25}$$

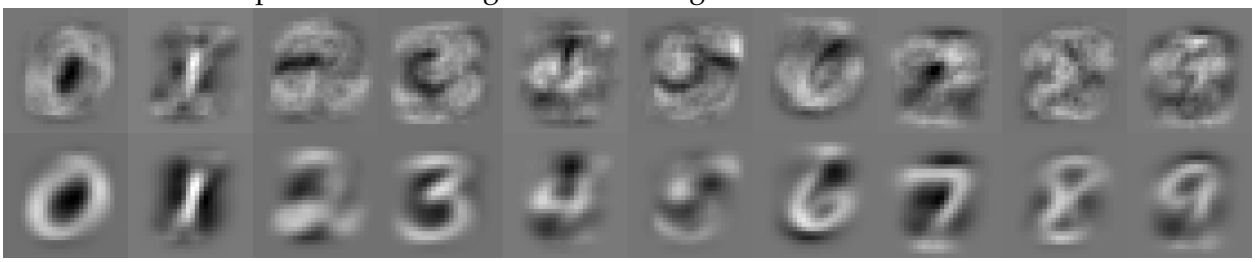
$$= \frac{1}{N} \sum_{n=1}^{N} \left[-x^n \left(y^n - \hat{y}^n \right) \right] + \lambda \frac{\partial w^{\mathrm{T}} w}{\partial w} \tag{26}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \left[-x^n \left(y^n - \hat{y}^n \right) \right] + 2\lambda w \tag{27}$$

Task 4b)

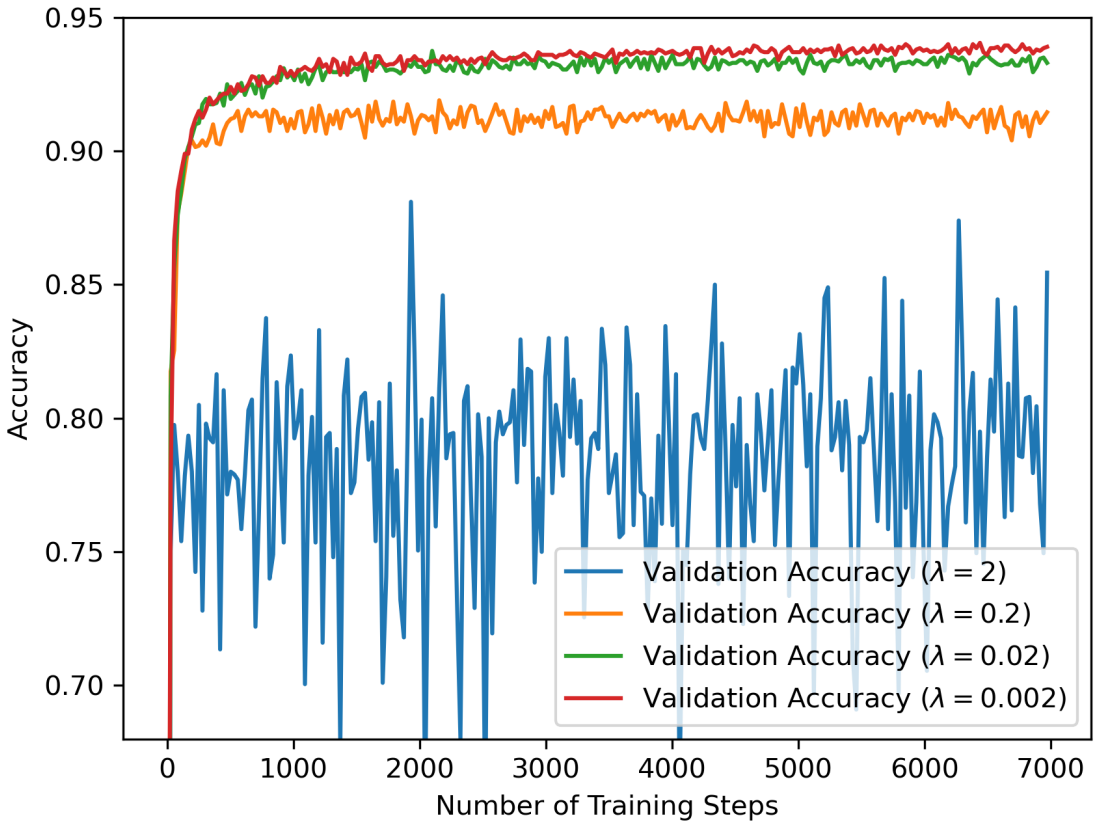
Using regularization, we get a model that is penalised to have smaller weights overall, leading to the scenario where the weights cannot overfit to niche features of the dataset subset of the domain. The model is forced to be more generalised in its weight representation, which in effect reduces dataset specific "noise" which is otherwise encoded into the weights of the network, as can be seen:

Top row: model weights when $L2$ regularization factor $\lambda = 0.0$



Bottom row: model weights when $L2$ regularization factor $\lambda = 2.0$

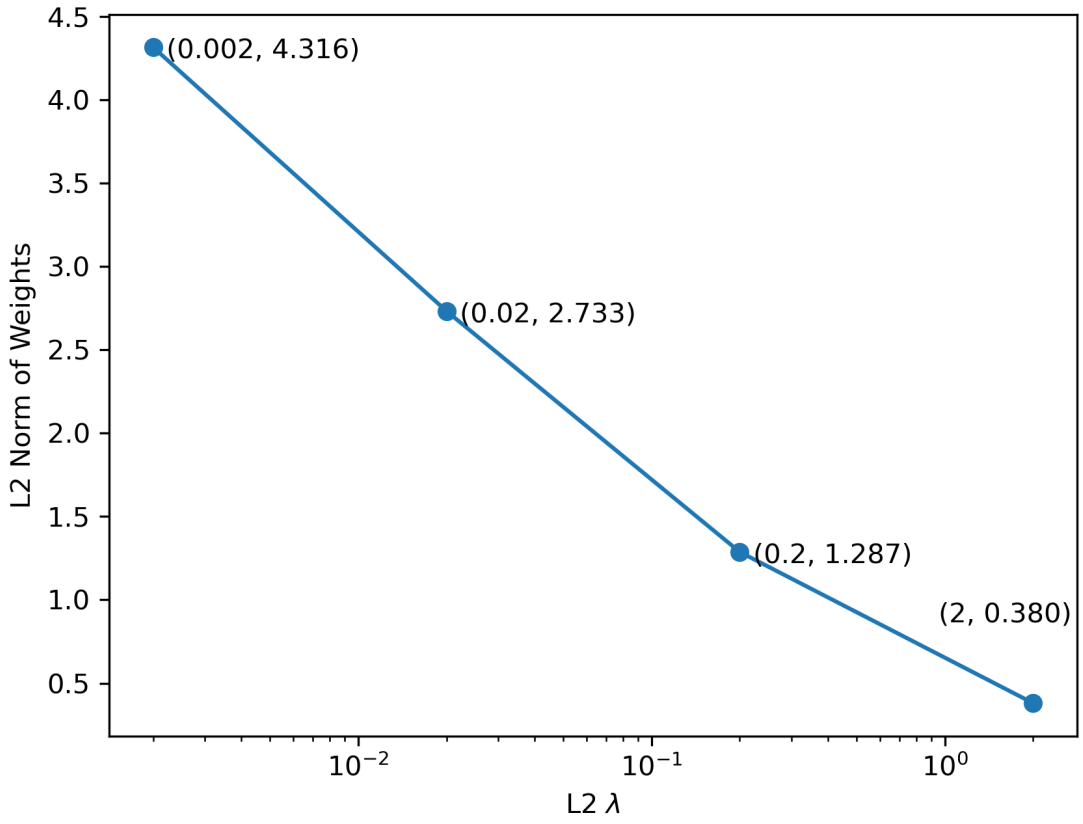
Task 4c)



Task 4d)

While regularization is applied to help the model generalize, it also forces the model to work within tighter constraints, meaning that for any given set of data, we'll be forcing the model to limit how it uses the data. The higher the degree of the regularization factor is, the less of the extremities of the dataset can be encoded within the weights. That is, the further we restrict the model to generalise, the worse it'll perform in outlier cases of data. This is known as the *bias-variance tradeof*. Increasing λ is essentially equivalent to reducing variance and increasing bias.

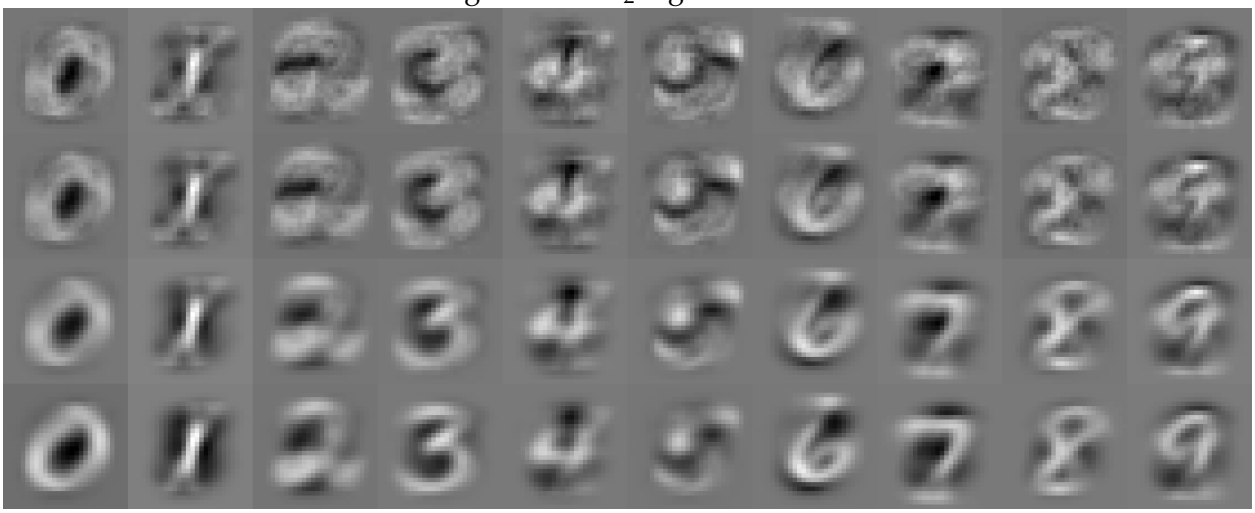
Task 4e)



We observe that the L_2 norm of the weights is approximately inversely proportional to λ.

Appendix

Row 1: model weights when L_2 regularization factor $\lambda = 0.002$
Row 2: model weights when L_2 regularization factor $\lambda = 0.02$



Row 3: model weights when L_2 regularization factor $\lambda = 0.2$
Row 4: model weights when L_2 regularization factor $\lambda = 2.0$

Figure 3: Weights of four configurations using L_2 regularization. The reduction of noise is clearly visible.