The code was written successfully, and tested thoroughly for bugs, however I was unable to accurately reproduce the results that were presented in the assignment description. More on this later.

The data loaded by the script was manipulated to be zero-mean, and was also divided by the standard deviation of the set (and not 255) as described in the assignment 2 pdf. And for all experiments in this assignment, the random seed was not fixed, and data was shuffled between each epoch during training.

1 Correctness of analytical gradient

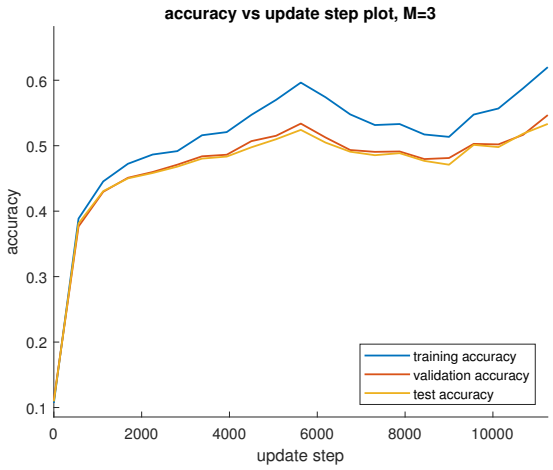
To verify that the analytical gradient was indeed correct, the analytically calculated gradient was compared against the numerically computed gradient, using the method

$$\text{alignment} = \frac{|g_a - g_b|}{\max(1e-6, |g_a| + |g_b|)} \tag{1}$$

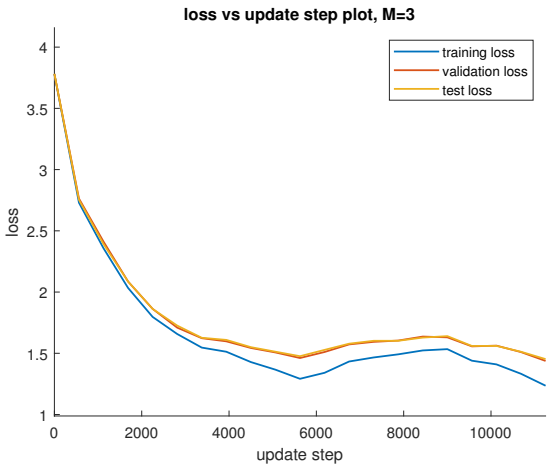
as in the previous two assignments. The numerical gradient code was slightly modified to account for a modified return and function signature of ComputeCost. The alignment between the numerical and analytical gradient was around 1e-8 or 1e-9 for all both W and b gradients with batch normalisation deactivated. With BN activated, all numerical gradients checked out (W, γ, β), except for the b_l layers (except for $k = l$, which checked out.). This was expected, because the batch normalisation step results in the b gradients to be trained to zero. The numerical gradient represented this as $1e-17-1e-18$, while the numerical gradient had some zeros, and some larger numbers than the anaklytical gradient (but still close to zero), which resulted in a poor alignment rating.

The alignment checks were performed using the 10 first full data samples from the `data_batch_1` set, with a 3 layer network with $[50, 50]$ and $[50, 30, 20, 20, 10, 10, 10, 10]$ layers respectively. The regularisation parameter λ was set to 0 and and 0.01 for the tests. The gammas were initialised to vectors of ones with M length (number of hidden nodes for that layer), and the betas to zeros. For this alignment test He initialisation was used.

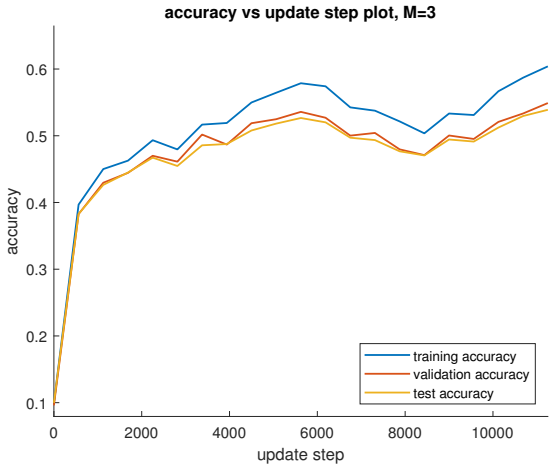
2 3 layer network, batch normalisation: yay or nay?



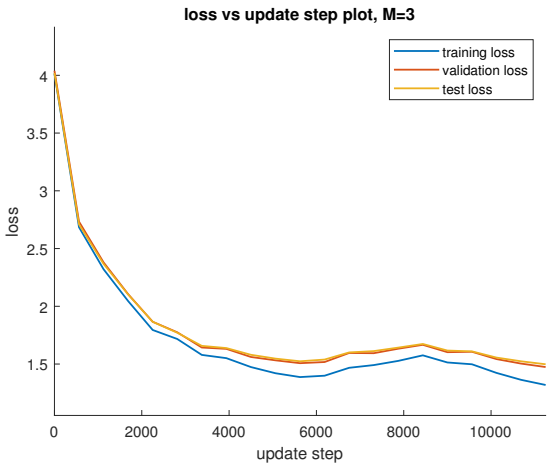
(a) Accuracy for 3 layer network with BN.



(b) Loss for 3 layer network with BN.



(c) Accuracy for 3 layer network without BN.



(d) Loss for 3 layer network without BN.

Figure 1: Accuracy and loss plots for 3 layer network with and without batch normalisation. Parameter settings can be found in Table 1.

The plots look as expected. However the resulting test accuracy was not as expected. See the next page for details.

Acknowledgment of error

For the previous two tasks I received some odd results. The test accuracy of the 3 layer dropped from 53.97 % without batch normalisation to 53.55 % when batch normalisation was activated. Which is both contrary to what was described in the assignment PDF, and did not match the 53.8 % test accuracy that Josephine received. This might have been affected by the initialisation of gamma and beta.

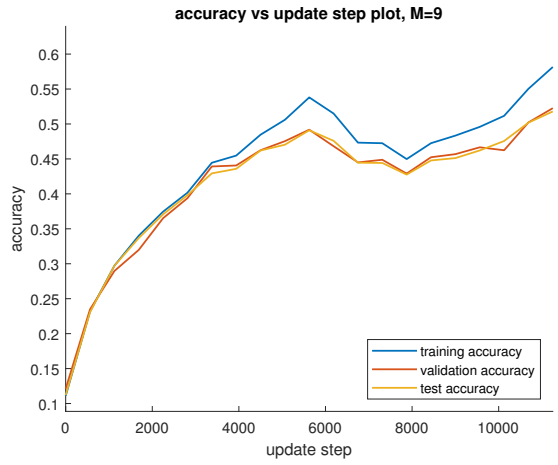
For the 9 layer network, the test accuracy using batch normalisation improved from 47.77 % (without) to 51.79 % (with), as expected.

Following discovering this bug, I have debugged my code for 3-4 full days, and even back-tracked all my source code to assignment 1 to make sure I did not have any bugs propagating through the assignments. After giving up, I even re-wrote all my code for this assignment from scratch, but still received the same results. I am to this day unsure what I did wrong, but cannot waste more time debugging this assignment, and hope that my effort put into this assignment is sufficient. I feel however, that through my debugging I have learnt sufficiently how batch normalisation works, which is the important thing.

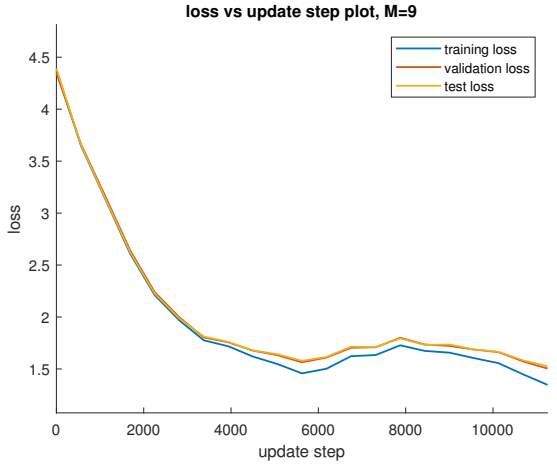
Table 1: Parameter settings for batch normalisation tests

Parameter	Value
Training set size	45000
Validation set size	5000
Test set size	10000
η_{max}	1e-1
η_{min}	1e-5
λ	0.005
# cycles (`n_cycles`)	2
batch size (`n_batch`)	100
cycle length (`n_s`)	2250
# epochs	20
γ_l initialisation	ones
β_l initialisation	zeros
W, b initialisation	He
α	0.9

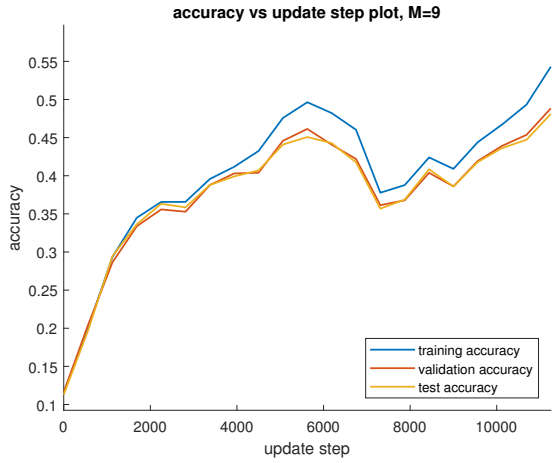
3 9 layer network, batch normalisation: yay or nay?



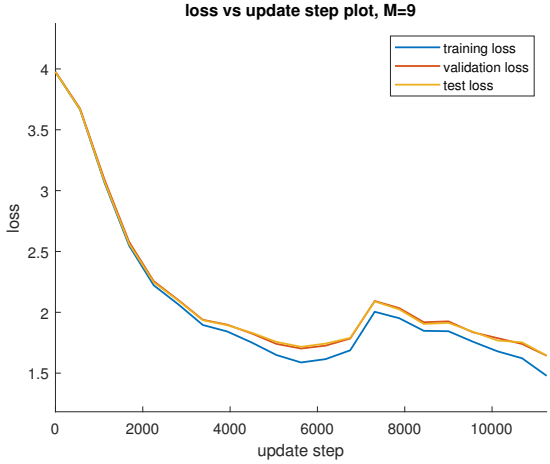
(a) Accuracy for 9 layer network with BN.



(b) Loss for 9 layer network with BN.



(c) Accuracy for 9 layer network without BN.



(d) Loss for 9 layer network without BN.

Figure 2: Accuracy and loss plots for 9 layer network with and without batch normalisation. Parameter settings can be found in Table 1.

The plots look good, and as expected the batch normalisation did help improve accuracy, from 47.77 % (without) to 51.79 % (with). Although this was a relatively small improvement, the He initialisation might have helped improve the training without batch normalisation as compared to using a poorer parameter initialisation.

4 Grid search to set λ

In exactly the same fashion as the previous assignment; to do the coarse grid search for the regularisation parameter λ, the training algorithm was run on 50 different λs, and then the best validation accuracy corresponding to the λ was saved. The λs were generated using the `logspacel_min, l_max`, where in my case for the coarse search $l_{min} = 10^{-5}$, $l_{max} = 0.1$. For this task the parameter settings in Table 1 were used, with exception for the varied λ.

Just as in the previous task, the range of "good" lambdas were in the 1e-3 range, so for the fine search, a linspace between 0.001 and 0.008 were used. The "good" lambda found, 0.0057 is debat-

able whether it is the best or now, because the random initialisation at the beginning of each round affects the performance of the network ever so slightly so that most lambda in this case for the 3 layer network, between 0.003 and 0.006 might be "equally as good".

The best test accuracy for this lambda was 53.88 % after 4 cycles. The fact that the found "good" lambda was in the same range as the previous assignment was hardly surprising, and the randomness in the initialisation and shuffling during MiniGD plays a larger role in the small performance differences.

5 Fine search to set λ

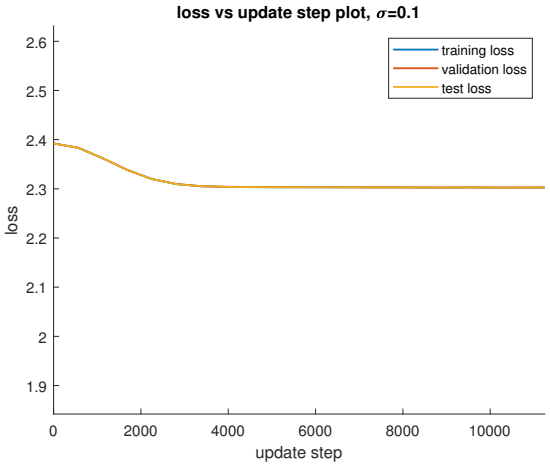
6 Sensitivity to initialisation

The experiments run in this task were done using the same parameter settings as earlier, as seen in Table 1.

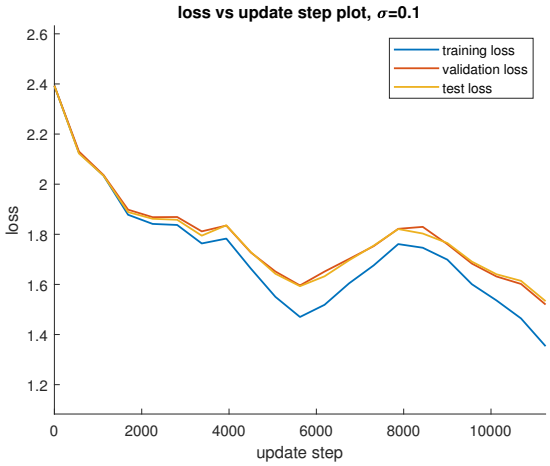
As expected, the batch normalisation improved the accuracy of the networks with normally distributed parameter initialisations with the same variance at each layer. However, I was suprised that the accuracy when not using batch normalisation was constantly capped at 10 %. After running the debugger, and looking at the S and X (H) matrices layer-by layer, it became evident that the values of the S and X matrices seemed to converge row-wise. As seen in 4, all the values row-wise are roughly the same, and after appling the SoftMax (which will be the same for every column), the best class will be # 9 for every column (sample), so if one assumes that the number of true samples per class is 1/10 * size of dataset, then it is natural that the accuracy is capped at 10.0 %. However I find that it is peculiar that the algorithm behaves this way, and would love to have an explanation for it, especially since the loss plots look so flat, as opposed to the ones for when using batch normalisation, whose appearance are as expected.

What is also even stranger, is that my algorithm gives relatively good results for He and Xavier initialised parameters W and b, and training without batch normalisation. And the fact that batch normalisation made the normally disributed initialisations with fixed variance (layer-wise), a lot better, as seen in the right column in Figure 3. So my batch normalisation algorithm obviously works, and since my algorithm without BN works for He and Xavier, that one surely works as well. I simply cannot explain the behaviour as seen in the left column of 3, because other people I have talked to have gotten relatively good results without batch normalisation for say $\sigma = 0.001$ while I haven't.

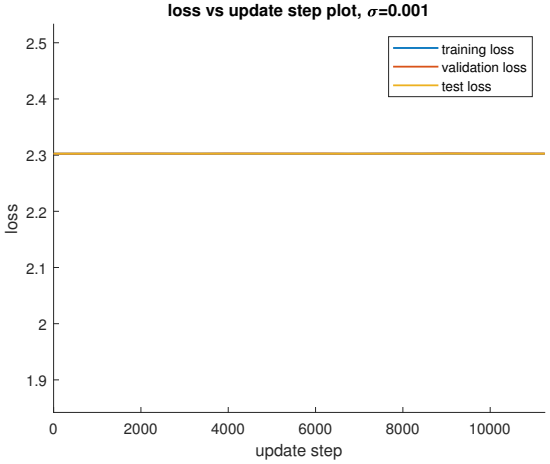
Notably for this experiment, is that the values in the S and X matrices started to became very small for $\sigma = 0.0001$, and matlab approximated e^{1e-8} to 1, so numerical precision might have played in here too (I did use double precision).



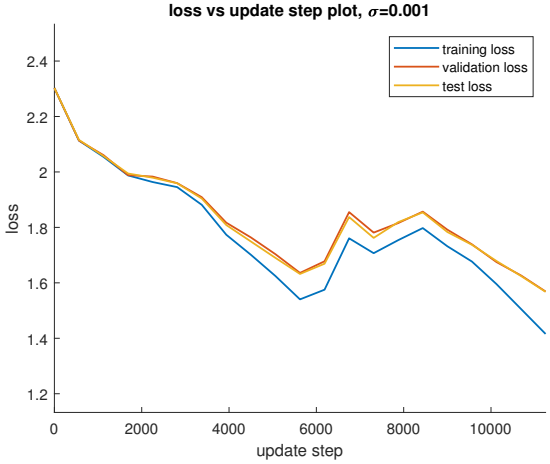
(a) Loss for 9 layer network without BN, $\sigma = 0.1$.



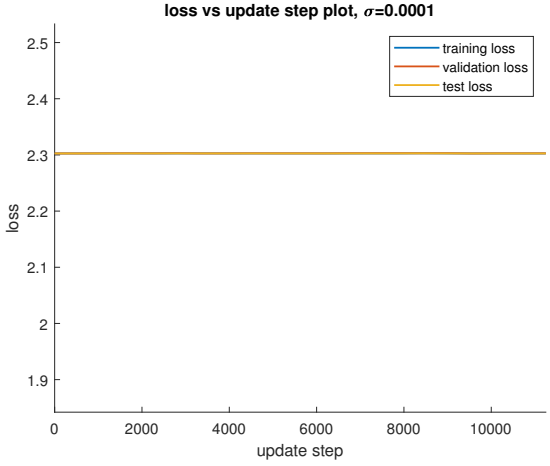
(b) Loss for 9 layer network with BN, $\sigma = 0.1$.



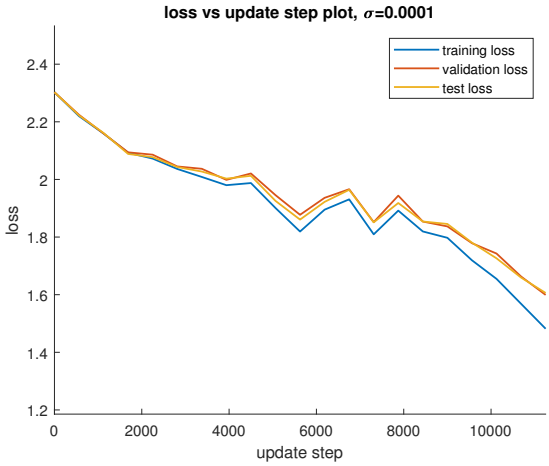
(c) Loss for 9 layer network without BN, $\sigma = 0.001$.



(d) Loss for 9 layer network with BN, $\sigma = 0.001$.



(e) Loss for 9 layer network without BN, $\sigma = 0.0001$.



(f) Loss for 9 layer network with BN, $\sigma = 0.0001$.

Figure 3: Accuracy and loss plots for 9 layer network with and without batch normalisation for different values of σ, the standard deviation of the normal distribution used to initialise the weight matrices W_l, and bias vectors b_l. Other parameter settings can be found in Table 1.

−0.00503373842179601	−0.00503373842172945	−0.00503373842093622
0.00212037695870497	0.00212037695869606	0.00212037695907540
−0.00198165598871020	−0.00198165598878489	−0.00198165599030538
−0.0129296544664732	−0.0129296544663863	−0.0129296544667471
−0.00192149355680394	−0.00192149355670894	−0.00192149355598630
−0.00551910519374485	−0.00551910519368041	−0.00551910519378684
−0.00729496110034918	−0.00729496110054113	−0.00729496110188688
0.00654406863826766	0.00654406863832342	0.00654406863813921
0.0173812070818205	0.0173812070819018	0.0173812070832210
0.00206107944730214	0.00206107944717431	0.00206107944628117

Figure 4: First 3 columns of the S matrix on the ninth layer using the initialised parameters W and b with $\sigma = 0.1$ prior to applying SoftMax.