DD2424 Deep Learning in Data Science: Assignment 3 (Opt.1)

Léo Noharet - lnoharet@kth.se

May 6, 2022

Note: For this assignment, as the complexity is higher than previous assignments, I chose to restructure my code significantly for better readability and flexibility. For this reason it might seem very different to my previous assignments.

1 Analytic gradient computation and correctness.

To check that the gradient computations are correct, I firstly augmented the code for the numeric computations by translating the matlab code to python code. Then, similarly to previous assignments, I computed the relative error between analytical and numeric computations for all four gradients (W, b, gamma, beta), with eps set to 10^{-6}. I also compared the gradients computed numerically and analytically with the help of the assert_almost_equal(actual, desired, decimal) function from the numpy.testing python library. This function verifies that the elements of the matrices *actual* and *desired* satisfy the following: $abs(desired - actual) < 1.5 * 10^{-decimal}$. [Dev] I used this with the parameter decimal set to 8, which proves that the difference is small between the two gradient matrices.
I ran the gradient computations on both a two layer and a three layer network. The results of these can be seen in figure 1.

```
--- Checking gradient computations for 2-layered nn w/ BN ---
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
| Layer |      diff grad_W      |      diff grad_b      |    diff grad_gammas   |    diff grad_betas    |
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
|   1   | 2.133044290911461e-09 | 2.220446090885151e-05 | 7.714980547835678e-10 | 7.661776409687037e-10 |
|   2   |  6.41045112246274e-10 | 2.724448417017184e-10 |          None         |          None         |
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
--- Checking gradient computations for 3-layered nn w/ BN ---
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
| Layer |      diff grad_W      |      diff grad_b      |    diff grad_gammas   |    diff grad_betas    |
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
|   1   | 2.0809187685549738e-09 | 2.4638918194910675e-11 | 1.7107740491624812e-09 | 2.059587387933357e-09 |
|   2   | 1.7990350297919796e-09 | 3.8483549122211534e-11 | 9.418002694982601e-10 | 7.726129224854142e-10 |
|   3   | 7.317882736022252e-10 |  4.84675609200439e-10 |          None         |          None         |
+-------+-----------------------+-----------------------+-----------------------+-----------------------+
```

Figure 1: Gradient computations for a 2-layer network (upper) and 3-layer network(lower)

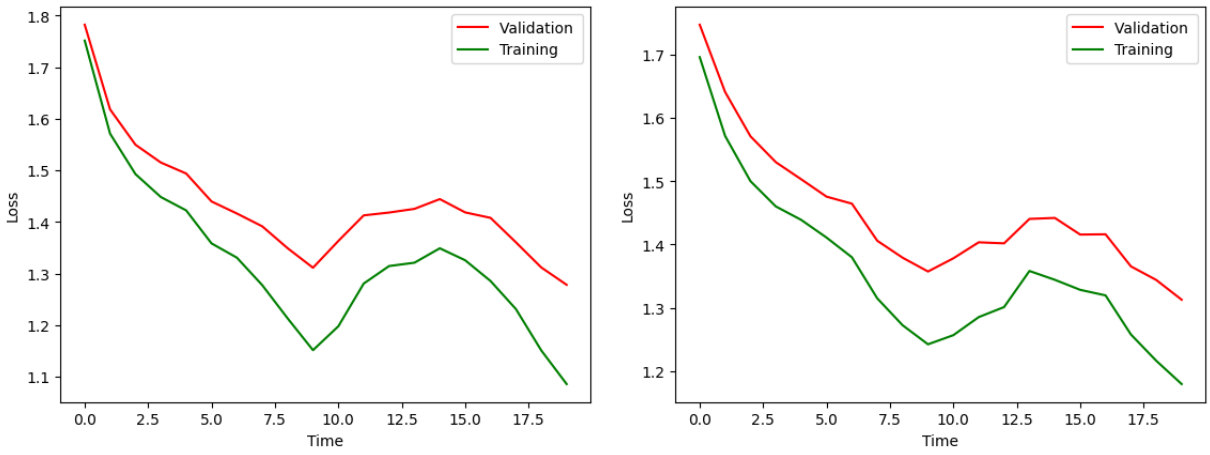

Figure 2: Loss function of 3 layer network with BN (left) and without BN (right) with with default parameters: 45000 training examples; He parameter initialization; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=5*450, cycles = 2.

2 Evolution of loss function on 3-layer network with and without BN given default parameter setting.

For the 3 layer network, given the default parameter setting given in the assignment description with batch randomization at the start of each epoch, a test accuracy of 52.9% was achieved without batch normalization (BN) and 53.45% with batch normalization which does not represent a significantly large improvement. The evolution of the loss function can be seen in figure 3

3 Evolution of loss function on 9-layer network with and without BN given default parameter setting.

For the 9 layer network, given the default parameter setting given in the assignment description with batch randomization at the start of each epoch, a test accuracy of 42.86% was achieved without batch normalization and 51.42% with batch normalization which is a significant difference as was expected. The evolution of the loss function can be seen in figure 3

4 Lambda search results

For all of the training done during the lambda search, the default parameters were used: 45000 training examples; He parameter initialization; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=5*450, cycles = 2 with batch randomization at the start of each epoch. For the coarse search, I trained the network with the following lambda values: 10^{-5}, $10^{-4.5}$, 10^{-4}, $10^{-3.5}$, 10^{-3}, $10^{-2.5}$, 10^{-2}, $10^{-1.5}$.

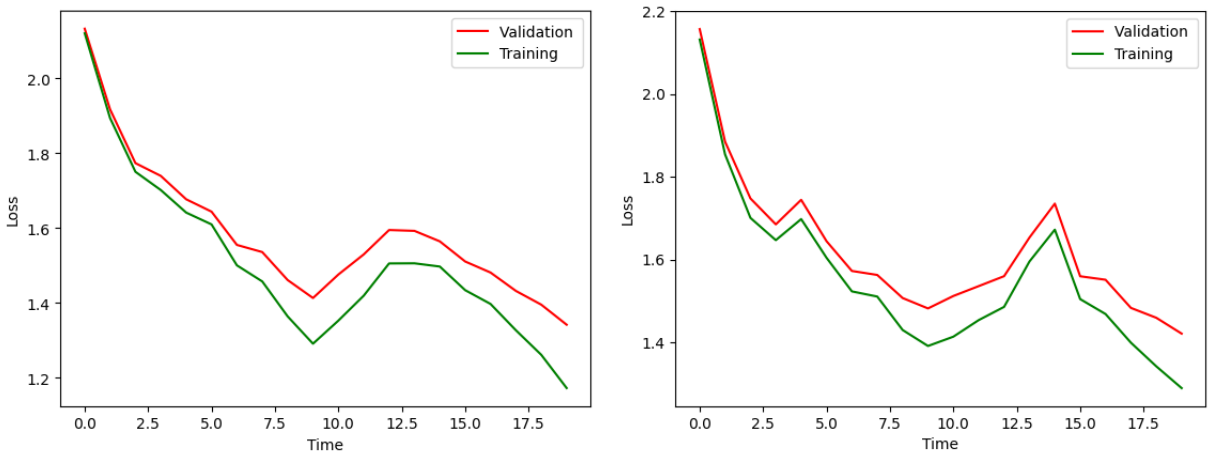

Figure 3: Loss function of 9 layer network with BN (left) and without BN (right) with default parameters: 45000 training examples; He parameter initialization; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=5*450, cycles = 2.

For the fine search, I ran the training for 30 different values of lambda. Similarly to the lambda search of assignment 2, these 30 values were randomly selected between the lambda value from the coarse search yielding the best validation accuracy and the lambda value giving the worst validation accuracy. range with the use of $lambda = 10^l$ where $l = l_min + (l_max - _min) *$random.uniform(0,1). to finally obtain lambda = 10e-2.29795015 which had an accuracy of 55.2% on the validation data.

Training a 3 layer network with this lambda value yielded a test accuracy of 53.98 % which is a slight improvement but not significant. The found lambda value 10e-2.29795015 is relatively close to the lambda value given in the default parameters from the assignment description.

5 Sensitivity to initialization

From the sensitivity to initialization experiment, we can see that batch normalization yields a more stable training since training is not effected as much by the initialization of the weights matrix. While the initialization of the weights matrix can affect the test accuracy of the network by several percentages, batch normalization helps to stabilise this significantly. When shifting the sigma values as instructed in the assignment, the loss function of the network using batch normalization is much less "chaotic" and minimizing than the loss function of the network not using batch normalization. For sigma = 1e-3 and 1e-4, the loss function is not decreasing. Further, the test accuracy of the BN-utilizing network is stable around 52% whilst the network not using BN yields a test accuracy of 10% for sigma = 1e-3 and 1e-4. Interestingly, the network not using BN performs rather similarly to the network using BN when sigma = 1e-1. This might be because when sigma = 1e-1, it corresponds approximately to using He or Xavier initialization. Therefore, we can see that when using BN, the method of weight initialization does not matter as much, which gives more stable training, whilst cor-

rect weight initialization is more important when not using BN. The loss functions of the experiment can be seen in figures 4, 5 and 6.

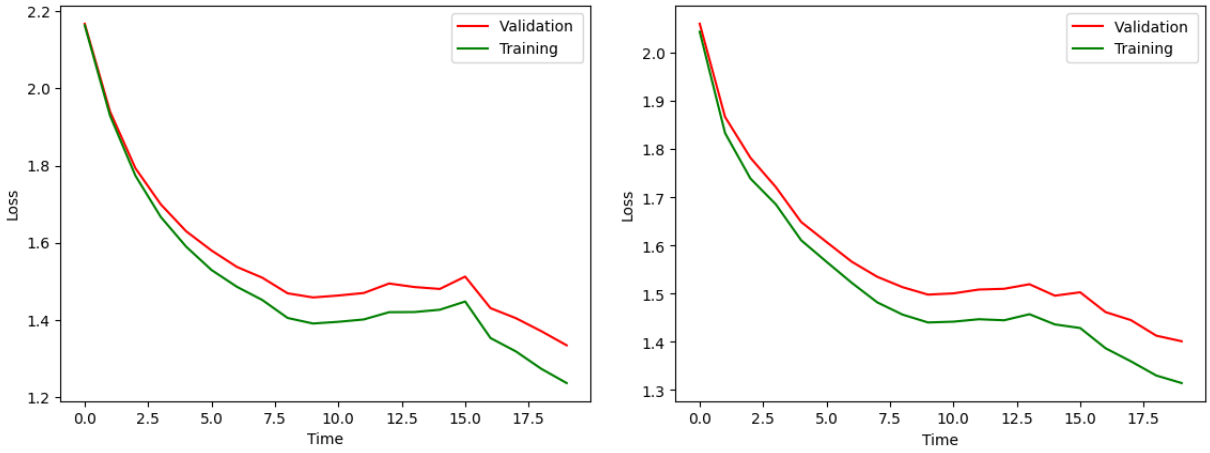

Figure 4: Loss function of 3 layer network with BN (left) and without BN (right) with default parameters: 45000 training examples; Sigma parameter initialization with sig=0.1; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=2*450, cycles = 2.

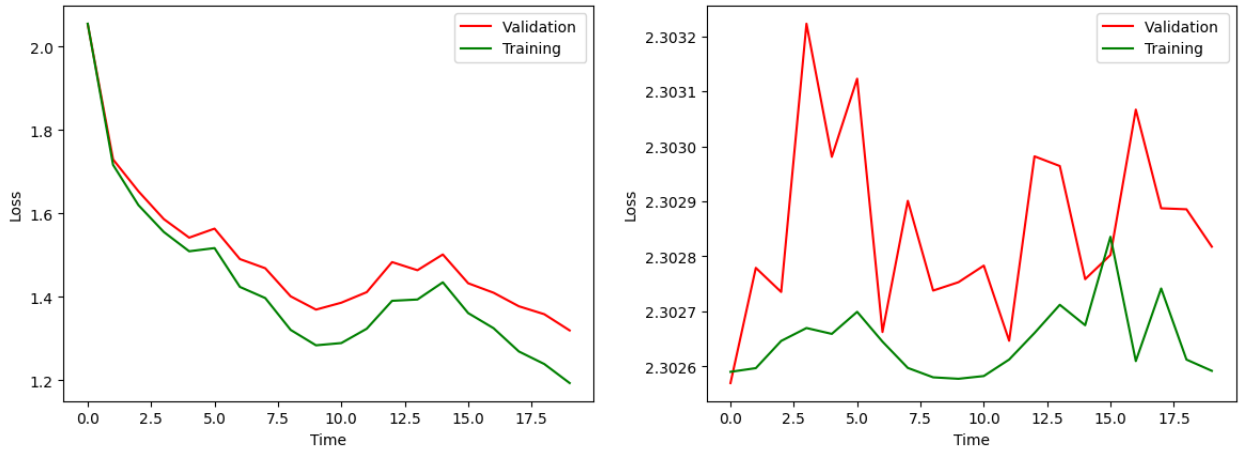

Figure 5: Loss function of 3 layer network with BN (left) and without BN (right) with default parameters: 45000 training examples; Sigma parameter initialization with sig=0.001; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=2*450, cycles = 2.

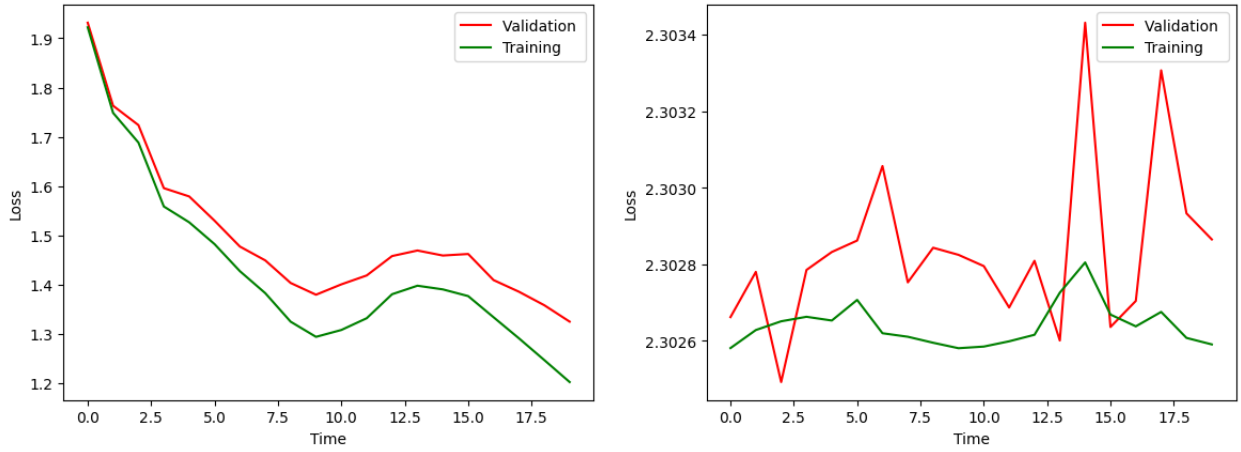

Figure 6: Loss function of 3 layer network with BN (left) and without BN (right) with default parameters: 45000 training examples; Sigma parameter initialization with sig=0.0001; lambda=0.005; eta min = 1e-5; eta max = 1e-1; and n_s=2*450, cycles = 2.

References

[Dev] Numpy Developers. numpy.testing.assert_almost_equal. https://numpy.org/doc/stable/reference/generated/numpy.testing.assert_almost_equal.html. Accessed: 2022-04-04.