Henrik Holm henholm@kth.se 2020-04-25

DD2424 Deep Learning in Data Science

Assignment 3 – base version

The toolset used for solving the assignment included default, built-in Python as well as the two external Python libraries numpy (for linear algebra purposes) and matplotlib (for plotting needs). The data set used was the CIFAR-10 collection of labelled images. A random seed of 12345 was used throughout the assignment.

Gradients without batch normalization

For ensuring that the implemented compute_gradients function (which computes gradients analytically) yielded the correct values, a test was devised. The test compares the results of compute_gradients with the results of compute_gradients_num, which calculates the same gradients, however using a significantly slower but more precise numerical approach. The test was conducted for a 2-layer network, a 3-layer network and a 4-layer network respectively. The test utilized numpy's allclose-function, which compares the values of two equally shaped matrices and returns either True or False depending on whether the compared elements are all close in value or not. With an absolute tolerance of 1e-07, the function returned True when comparing computed analytical gradients to computed numerical gradients for a subset consisting of 10 images from the CIFAR-10 data, meaning that all elements of the gradients were similar. This was the case for both the 2-layer, the 3-layer and the 4-layer network. Thus, the analytical gradient computations were deemed bug-free.

Exercise 2: Can I train multi-layer networks?

Exercise 2.1: replication of default results of Assignment 2

In this section, a 2-layer network with 50 nodes in the hidden layer was trained using the new K-Layer Network, for which the amount of layers could be freely specified during initialization. Results from initializing the new K-Layer Network to have 2 layers with 50 nodes in the hidden layer are shown in Figure 1 below. As can be seen, results are almost identical to results from the default network from Assignment 2. After reaching this point, the K-Layer Network was deemed to have been correctly implemented.

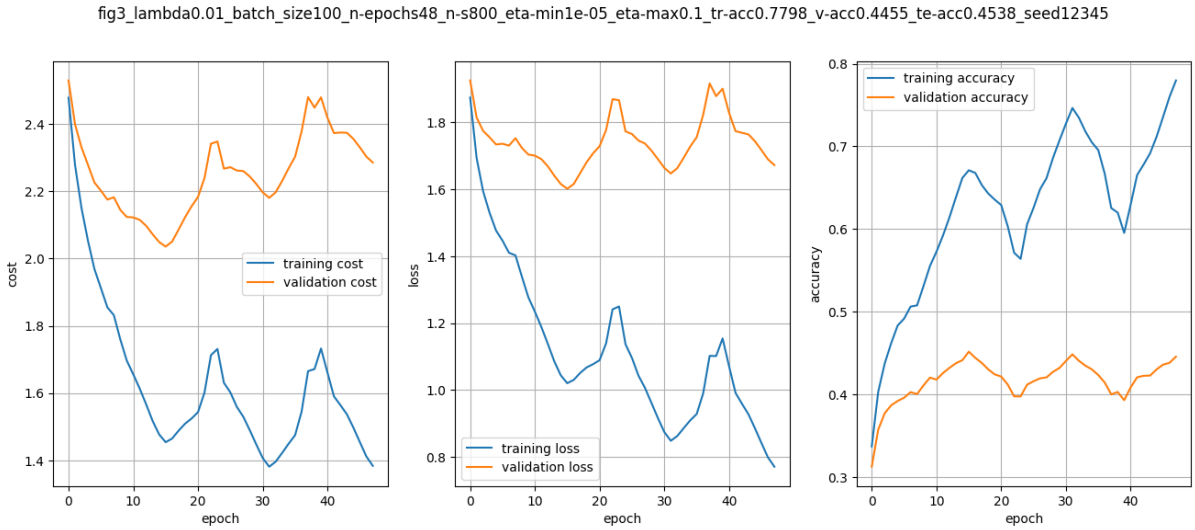

Figure 1: Cost plot, loss plot and accuracy plot respectively for three cycles of training, effectively replicating Figure 4 in the Assignment2 sheet. Here, lambda=0.01, batch_size=100, n_epochs=48, n_s=800, eta_min=1e-5 and eta_max=1e-1 were used. The final training accuracy equaled 77.98%, the final validation accuracy equaled 44.55% and the final test set accuracy equaled 45.38%.

Exercise 2.2: training 3-layer network with 50 and 50 nodes in first and second layer respectively

In this section, a 3-layer network with 50 and 50 nodes in the first and second hidden layers respectively was trained for two cycles. Parameters were initialized using He initialization. Results and the set of parameters (as specified in the assignment sheet) are displayed in Figure 2 below:

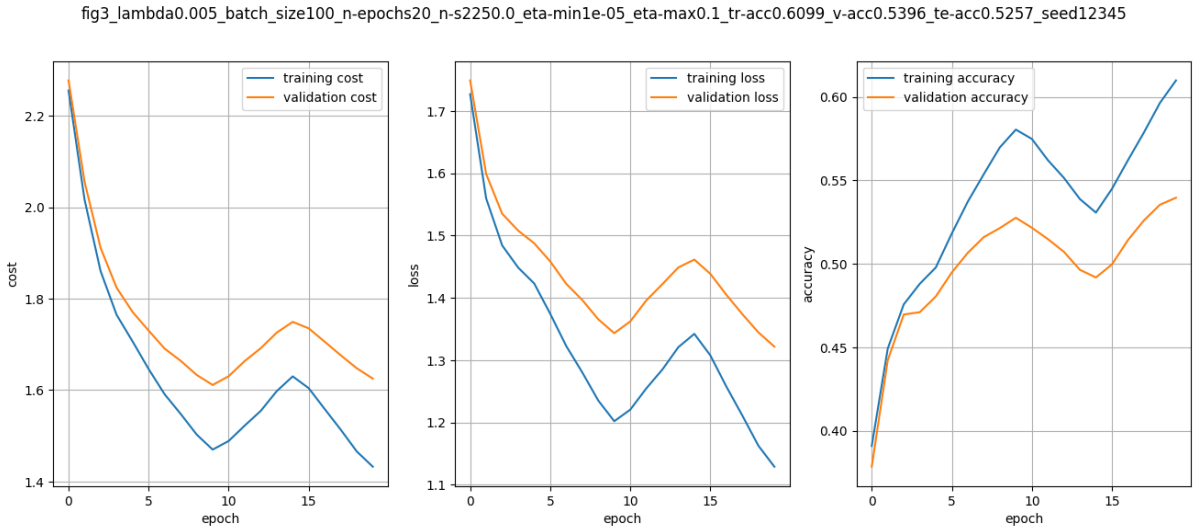

*Figure 2: Cost plot, loss plot and accuracy plot respectively for two cycles of training of a 3-layer network with 50 and 50 nodes respectively in the hidden layers. Here, lambda=0.005, batch_size=100, n_epochs=20, n_s=(5 * 45,000 / 100), eta_min=1e-5 and eta_max=1e-1 were used. The final training accuracy equaled 60.99%, the final validation accuracy equaled 53.96% and the final test set accuracy equaled 52.57%.*

The final test set accuracy was roughly equal to that specified in the assignment sheet (~52%).

Exercise 2.3: training 9-layer network

In this section, a 9-layer network with 50, 30, 20, 20, 10, 10, 10 and 10 nodes in the respective hidden layers was trained for two cycles. Parameters were initialized using He initialization. Results and the set of parameters (as specified in the assignment sheet) are displayed in Figure 3 below:

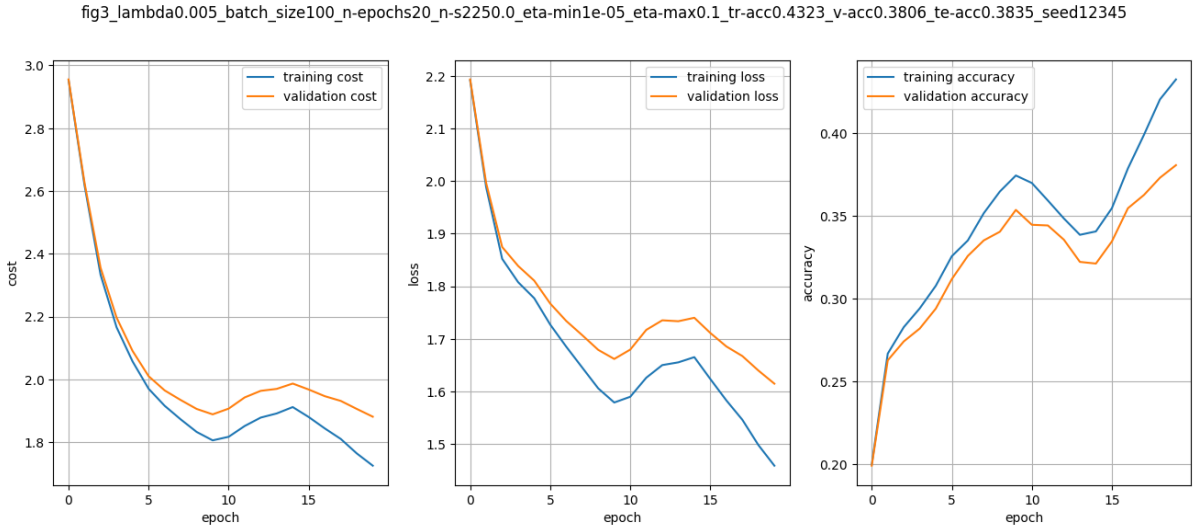

*Figure 3: Cost plot, loss plot and accuracy plot respectively for two cycles of training of a 9-layer network with 0, 30, 20, 20, 10, 10, 10 and 10 nodes in the respective hidden layers. Here, lambda=0.005, batch_size=100, n_epochs=20, n_s=(5 * 45,000 / 100), eta_min=1e-5 and eta_ma=1e-1 were used. The final training accuracy equaled 43.23%, the final validation accuracy equaled 38.06% and the final test set accuracy equaled 38.35%.*

As can be seen from Figure 3, the deeper network dropped significantly in performance compared to the previous two.

Exercise 3: Implement batch normalization

Exercise 3.0: Gradients with batch normalization

For ensuring that the implemented compute_gradients function (which computes gradients analytically) yielded the correct values, a test was devised. The test compares the results of compute_gradients with the results of compute_gradients_num, which calculates the same gradients, however using a significantly slower but more precise numerical approach. The test was conducted for a 2-layer network (with 50 hidden nodes) and a 3-layer network (with 50 and 50 hidden nodes respectively). The test utilized numpy's allclose-function, which compares the values of two equally shaped matrices and returns either True or False depending on whether the compared elements are all close in value or not. Our test also outputs the relative error of the two compared matrices.

For the 2-layer network and an absolute tolerance of 1e-07:

- Layer2: allclose returned True for the **bias**,

- Layer2: allclose returned True for the **gamma**,
- Layer2: allclose returned True for the **beta**,
- Layer2: allclose returned True for the **weights**,

- Layer1: allclose returned True for the **bias**,
- Layer1: allclose returned True for **gamma**,
- Layer1: allclose returned True for **beta**,
- Layer1: allclose returned False for the **weights**, and the relative error equaled ~1.

For the 3-layer network and an absolute tolerance of 1e-07:

- Layer3: allclose returned True for the **bias**. Relative error of 5.023e-08.
- Layer3: allclose returned True for the **gamma**. Relative error of 0.
- Layer3: allclose returned True for the **beta**. Relative error of 0.
- Layer3: allclose returned True for the **weights**. Relative error of 1.395e-07.

- Layer2: allclose returned True for the **bias**. Relative error of 2.162e-03.
- Layer2: allclose returned True for the **gamma**. Relative error of 1.480e-06.
- Layer2: allclose returned True for the **beta**. Relative error of 6.719e-07.
- Layer2: allclose returned False for the **weights**. Relative error of ~1.

- Layer1: allclose returned True for the **bias**. Relative error of 0.978.
- Layer1: allclose returned False for **gamma**. Relative error of ~1.
- Layer1: allclose returned False for **beta**. Relative error of ~1.
- Layer1: allclose returned False for the **weights**. Relative error of ~1.

One can clearly see that a discrepancy exists between the numerically and the analytically computed gradients. One can also see how errors from deeper layers snowball when propagated to shallower layers. It is possible that the initialization plays a role in the errors. I spent *significant* time and effort in trying to fix the analytical gradient calculations. In the end, I continued with the assignment, and realized that when training using my implementation, the acquired accuracy of the classifier was almost identical to the one stated in the assignment sheet. Moreover, I ran an experiment similar to the sanity check from assignment 2. In other words, I tested whether my implementation was capable of overfitting on a subset of 100 images. As can be seen in Figure 4, the classifier is capable of overfitting.

Already after about 50 epochs, the training loss is practically zero. This led me to deem my implementation to work. It is possible that it was my numerical computations that were wrong.

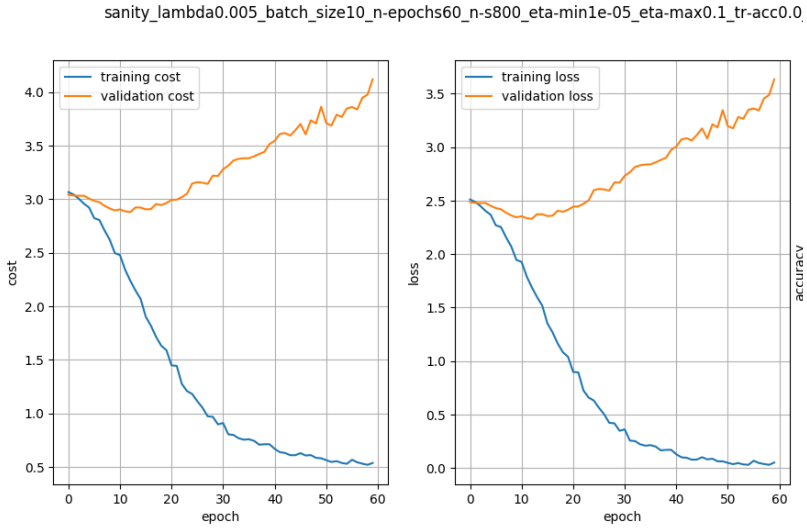

Figure 4: Sanity check. As can be seen, the training loss becomes zero after a sufficient amount of epochs, indicating that the classifier is capable of overfitting.

Training of 3-layer network with 50 and 50 nodes in the first and second hidden layers respectively

The exponential moving average of the batch mean and variances for the un-normalized scores for each layers (as defined by equations 38 and 39) was added to the mini-batch gradient descent algorithm. Random shuffling of training samples after each epoch was also added. Without performing a lambda search of any sort, the following results were achieved:

- Training set accuracy: 59.32%
- Validation set accuracy: 52.02%
- **Test set accuracy: 52.46%**

using 45,000 images for training and the hyperparameter settings: lambda=0.005, n_epochs=20, two cycles of training, batch_size=100, n_s=(5 * 45,000 / 100), eta_min=1e-05, eta_max=1e-01, alpha=0.9. Plots are shown in Figure 5 here below.

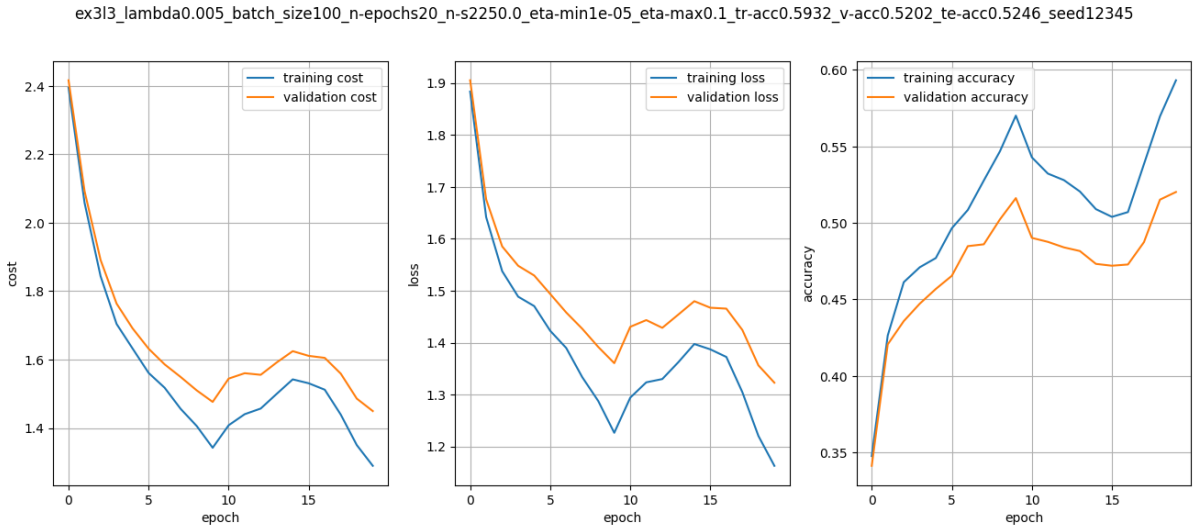

*Figure 5: Cost plot, loss plot and accuracy plot respectively after training for two cycles on 45,000 images. Here, lambda=0.005, batch_size=100, n_epochs=20, n_s=(5 * 45,000 / 100), eta_min=1e-5, eta_max=1e-1 and alpha=0.9 were used. The final training set accuracy equaled 59.32%, the validation set accuracy equaled 52.02% and the test set accuracy equaled 52.46%.*

Coarse lambda search: 3-layer network with 50 and 50 nodes in the 1st and 2nd hidden layers

8 values of the regularization term lambda were sampled uniformly from the range [1e-5, 1e-1]. Then, for each of these values, a training session was executed using 45,000 images for training and the hyperparameter settings: n_epochs=20, batch_size=100, n_s=(5 * 45,000 / 100), eta_min=1e-05, eta_max=1e-01, alpha=0.9 and batch_norm=True.

The results and plots from each session were stored locally. The results are displayed in the following table:

Lambda	Final validation acc.	Mean of top 5 val. acc.	Final training acc.
0.00001	51.76%	51.72%	62.16%
0.01429	51.56%	49.99%	56.49%
0.02857	50.62%	47.78%	53.46%
0.04286	49.62%	46.49%	52.22%
0.05714	48.42%	45.91%	51.24%
0.07143	47.9%	44.74%	50.27%
0.08571	47.5%	44.28%	49.19%
0.10000	46.38%	43.12%	48.98%

Based on the highest validation set accuracies obtained, a subset of lambdas was identified for the fine search. **This subset of lambdas were found in the range [0.00001, 0.02857]**, since it was between these values that the trained classifiers yielded the highest classification accuracies.

Fine lambda search: 3-layer network with 50 and 50 nodes in the 1st and 2nd hidden layers

Values for the fine search were now once again generated uniformly from the range identified in the previous step (as opposed to being uniformly drawn). The same hyperparameter settings were used. Again, 8 values were drawn, the best of which was:

- **lambda = 0.0041**, which yielded a final **validation set accuracy of 53.26%**

This value for lambda will be the one used in later sections of this assignment.

Training of 3-layer network with 50 and 50 nodes in the first and second hidden layers respectively and our "optimal" lambda of 0.0041

Using our semi-optimal lambda=0.0041, we then trained our 3-layer classifier using 45,000 images and tested it using the hyperparameter settings: n_epochs=30, 3 cycles of training, batch_size=100, n_s=(5 * 45,000 / 100), eta_min=1e-05, eta_max=1e-01, alpha=0.9 and batch_norm=True.

- **Test accuracy achieved: 53.68%**

Please refer to Figure 6 for corresponding plots.

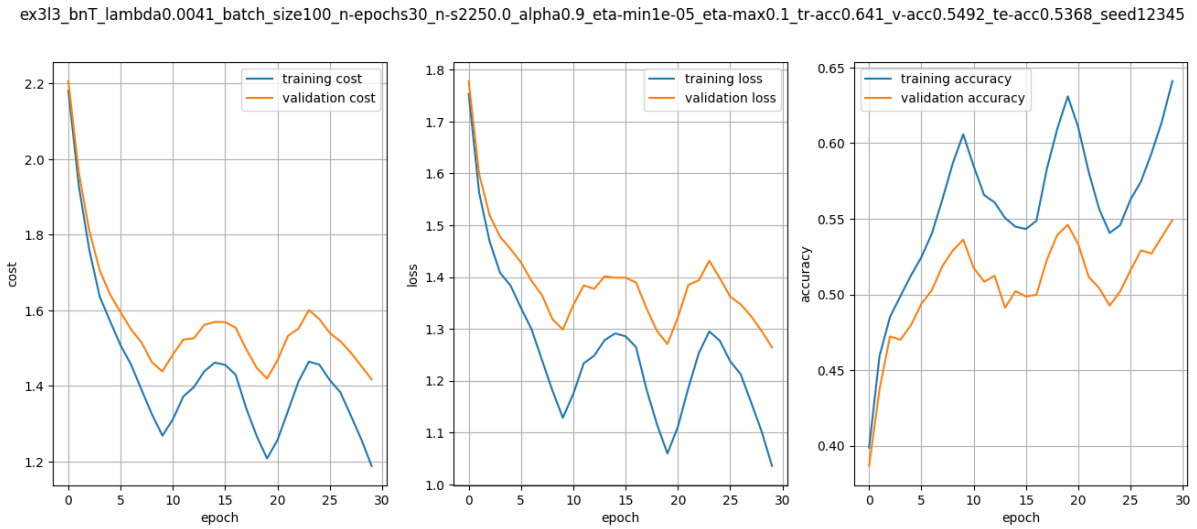

*Figure 6: Cost plot, loss plot and accuracy plot respectively after training for two cycles on 45,000 images. Here, lambda=0.0041, batch_size=100, n_epochs=30, n_s=(5 * 45,000 / 100), eta_min=1e-5, eta_max=1e-1 and alpha=0.9 were used. The final training set accuracy equaled 64.10%, the validation set accuracy equaled 54.92% and the test set accuracy equaled 53.68%.*

Training of 9-layer network with 50, 30, 20, 20, 10, 10, 10 and 10 nodes in the hidden layers and our "optimal" lambda of 0.0041

Using our semi-optimal lambda=0.0041, we now trained our 9-layer classifier using 45,000 images and tested it using the hyperparameter settings: n_epochs=30, 3 cycles of training, batch_size=100, n_s=(5 * 45,000 / 100), eta_min=1e-05, eta_max=1e-01, alpha=0.9 and batch_norm=True.

- **Test accuracy achieved: 52.45%**

which is a major improvement compared to when not using batch normalization.

Please refer to Figure 7 for corresponding plots.

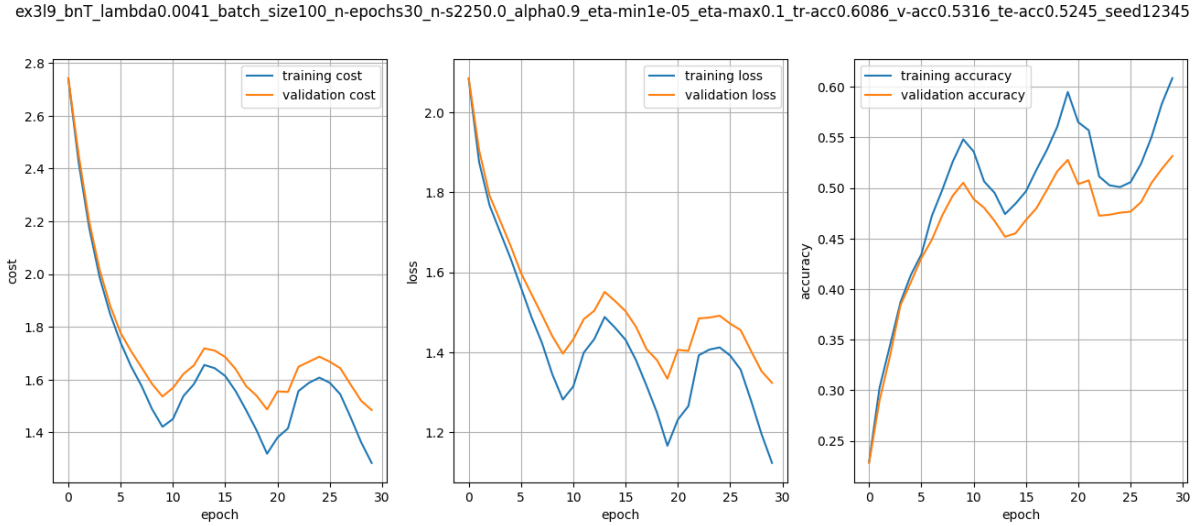

*Figure 7: Cost plot, loss plot and accuracy plot respectively after training for two cycles on 45,000 images using our 9-layer network. Here, lambda=0.0041, batch_size=100, n_epochs=30, n_s=(5 * 45,000 / 100), eta_min=1e-5, eta_max=1e-1 and alpha=0.9 were used. The final training set accuracy equaled 60.86%, the validation set accuracy equaled 53.16% and the test set accuracy equaled 52.45%.*

Sensitivity of initialization

Sigma	Without batch normalization	With batch normalization
1e-1	final test acc 53.52%	final test acc 53.68%
1e-3	final test acc 51.35%	final test acc 53.42%
1e-4	final test acc 10.00%	final test acc 53.40%

All of the below figures (Figures 8 – 13) use the same hyperparameters settings:

3-layer network with 50 nodes in each hidden layer, trained on 45,000 images, n_epochs=20, 2 cycles of training, batch_size=100, n_s=(5 * 45,000 / 100), eta_min=1e-05, eta_max=1e-01 and alpha=0.9

Only the boolean *batch_norm* and the *sigma* value used differs between the sessions.

One can conclude that deep neural networks are significantly less sensitive to the weight initialization scheme used when batch normalization is employed. In other words, one can say that batch normalization ensures the learning ability of a network, even when faced with possibly suboptimal weight initializations. For cases in which batch normalization was used (Figures 11, 12, 13), a cyclical behavior of the loss is clearly displayed. This is not the case for Figures 9 – 10, where batch normalization was not used. Here, the loss function displays a more linear behavior.

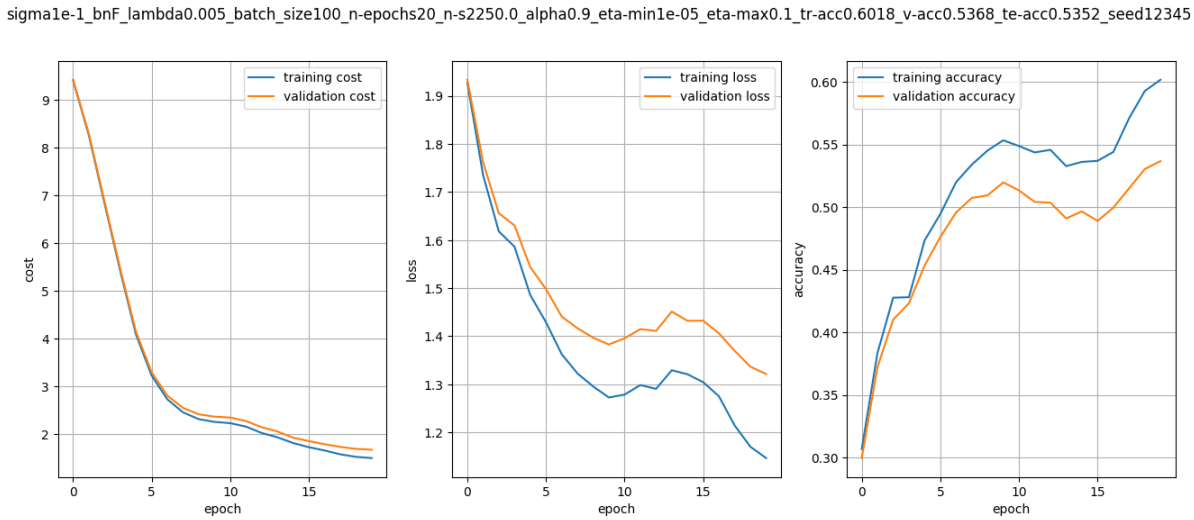

Figure 8: without batch normalization and with sigma = 1e-1.

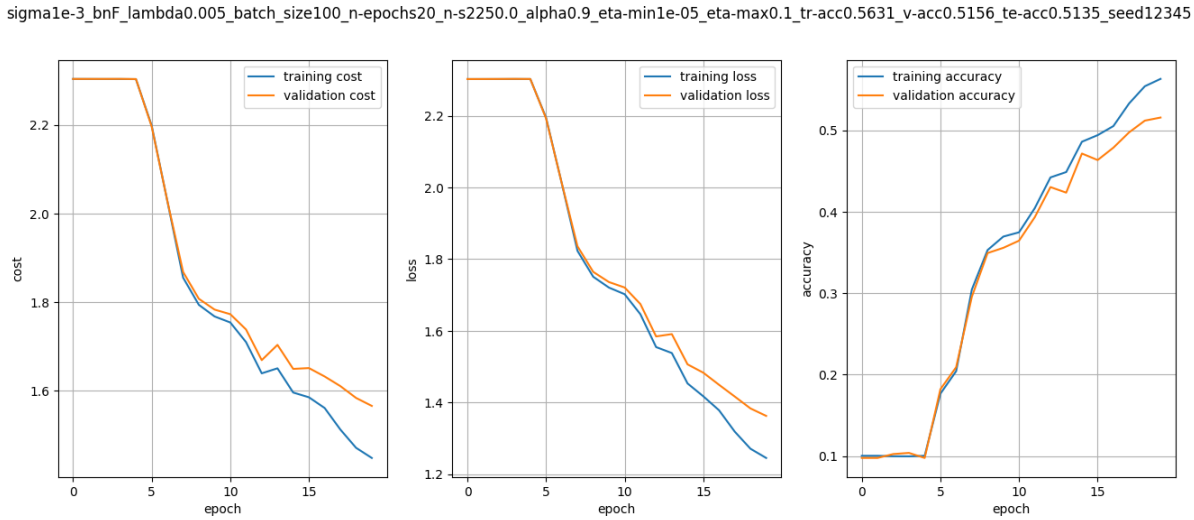

Figure 9: without batch normalization and with sigma = 1e-3.

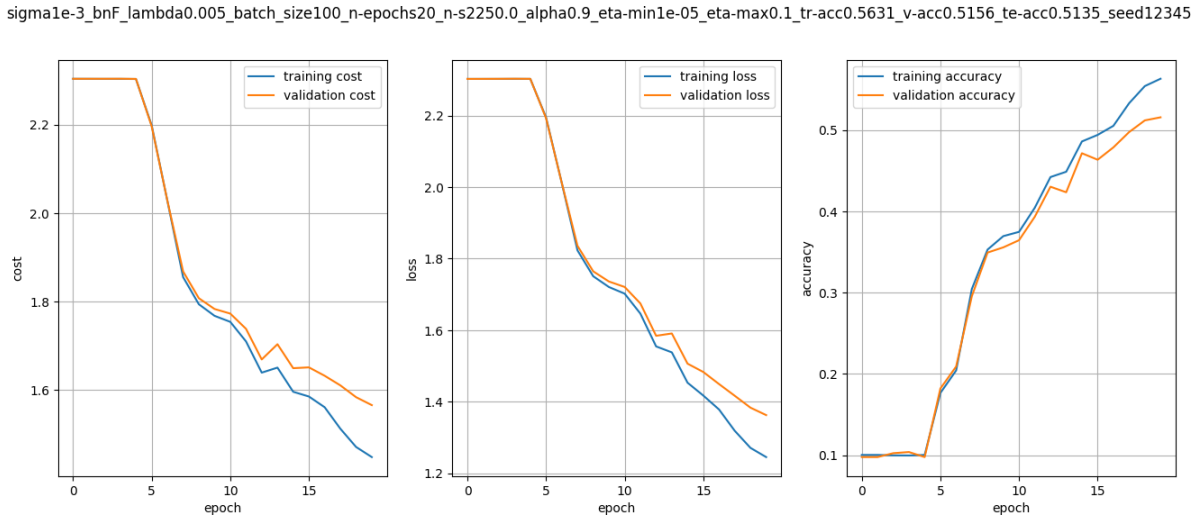

Figure 10: without batch normalization and with sigma = 1e-4.

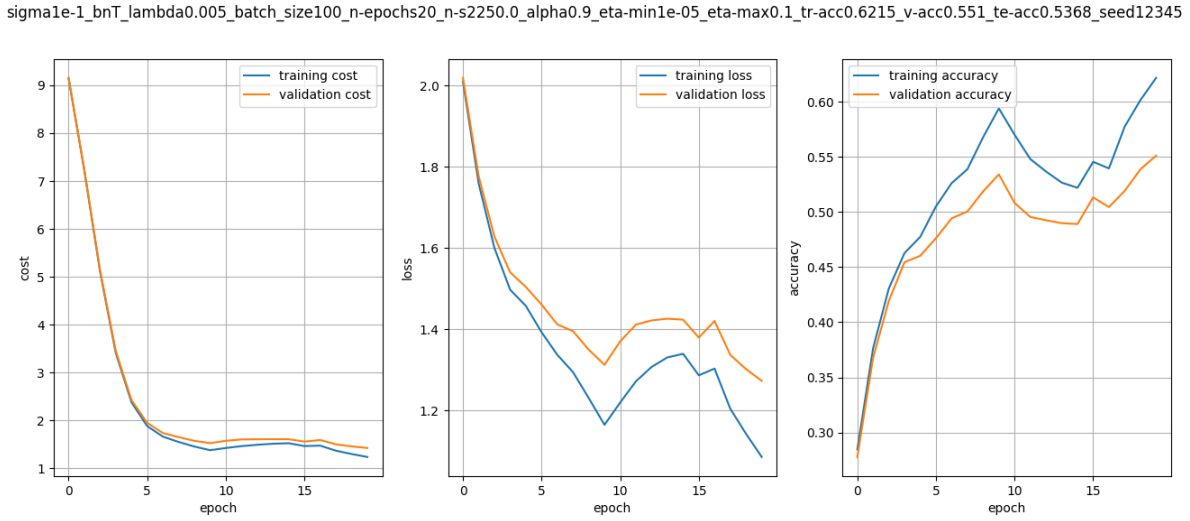

Figure 11: with batch normalization and with sigma = 1e-1.

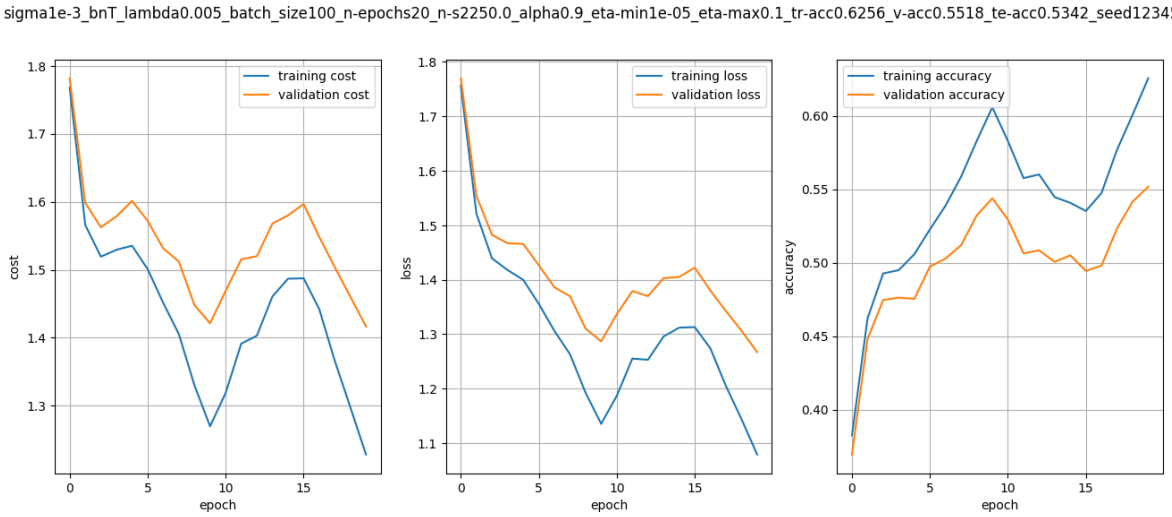

Figure 12: with batch normalization and with sigma = 1e-3.

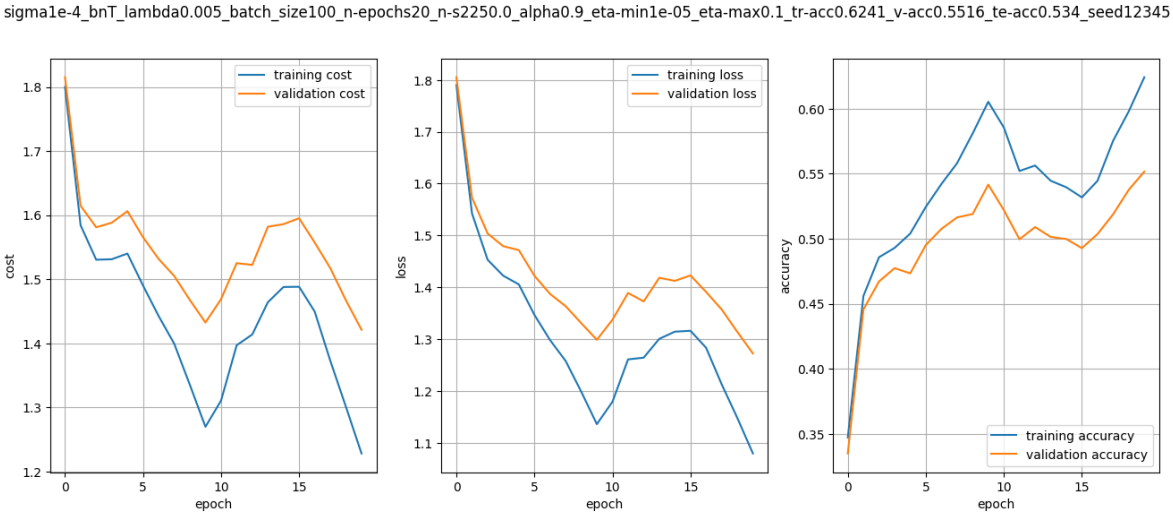

Figure 13: with batch normalization and with sigma = 1e-4.