Deep Learning in Data Science, DD2424

Short report on assignment 3

k-layer classification network for CIFAR-10 with Batch normalization

Simon Jarvers

May 8, 2023

1 Main objectives and scope of the assignment

My goals in the assignment were:

- to implement a k-layer classification network,
- to use the gradient decent method to train the network,
- to add Batch Normalization to the network,
- to train the network on the well-known CIFAR-10 data set.

To study the points of interest, the experiments and plots were created in Python using the widely known libraries, i. e. NumPy and Matplotlib. This report was written without the use of ChatGPT.

2 Classification with a k-layer classification network

Generalizing the 2-layer network from assignment 2 into a k-layer network motivated me to refactor my code and make it more generic. I introduced a base class `class Layer` from which the three classes `class FCLayer`, `class BNLayer` and `class ActivationLayer` are inherited. This also modularized the forward and backward pass. I tested the gradients before and after implementing Batch Normalization. The computations seem to be correct as illustrated in Figure 1 and 2.

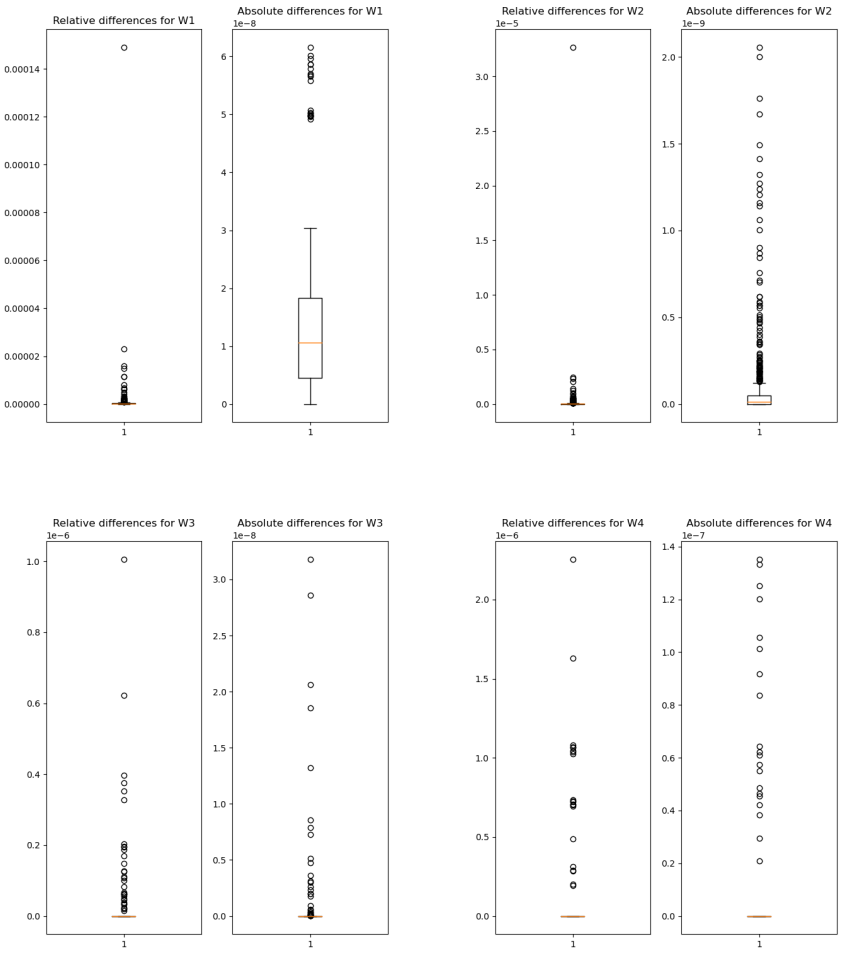

Figure 1: Box plots for W1, W2, W3 and W4 without BN

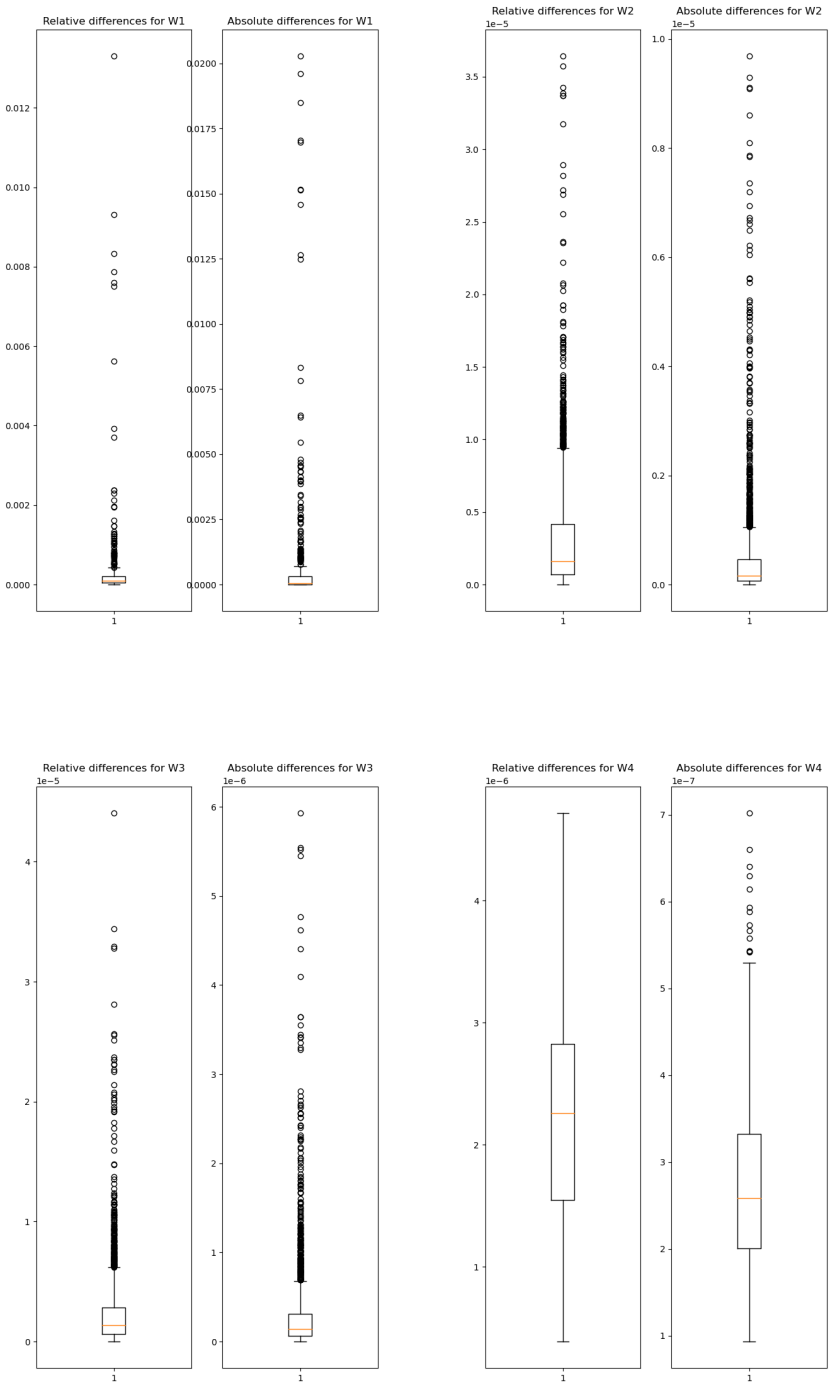

Figure 2: Box plots for W1, W2, W3 and W4 with BN

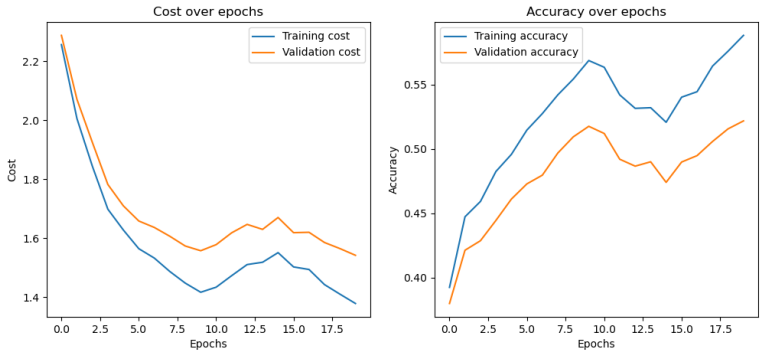

Figure 3: Cost function and accuracies without BN for 3-layer NN. Test accuracy was 52.80%

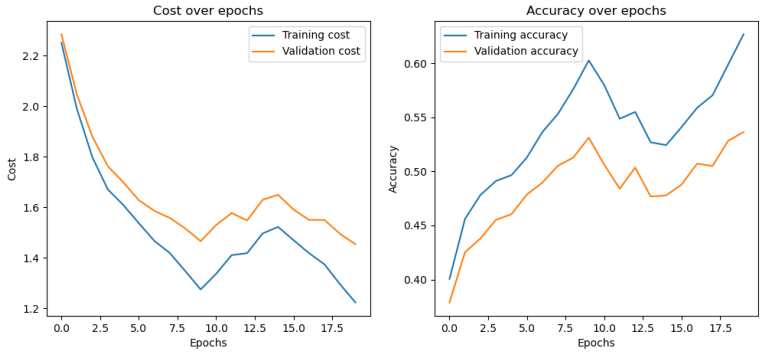

Figure 4: Cost function and accuracies with BN for 3-layer NN. Test accuracy was 53.89%

2.1 3-layer network

For the given default settings (2 x 50 unit hidden layer, $\lambda = 0.005$, cyclic learning rate as proposed in the assignment description) the 3-layer network with Batch Normalization performed only marginally better than the network without BN (1% increase) as shown in Figure 3 and 4. I think a 3-layer NN is not deep enough to intensively profit of Batch Normalization. This however changes when we look into deeper networks in the next section.

2.2 9-layer network

Moving on to the 9-layer NN Batch Normalization is now of great value. As the network gets deeper it becomes harder and harder to properly train it. This results in a decrease of the test accuracy to 41.74% whereas the network with BN achieves 52.20%. The training process can be seen in Figure 5 and 6.

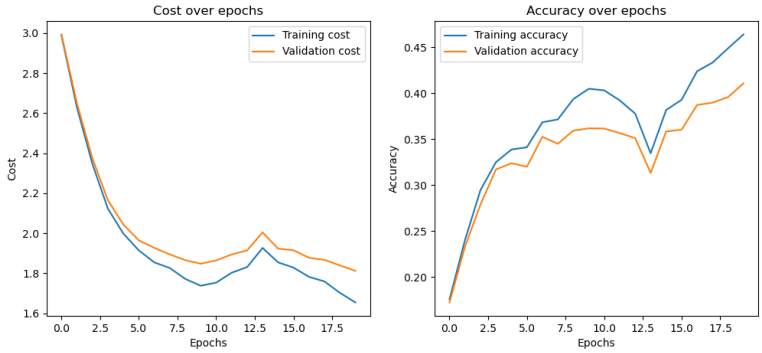

Figure 5: Cost function and accuracies without BN for 9-layer NN. Test accuracy was 41.74%

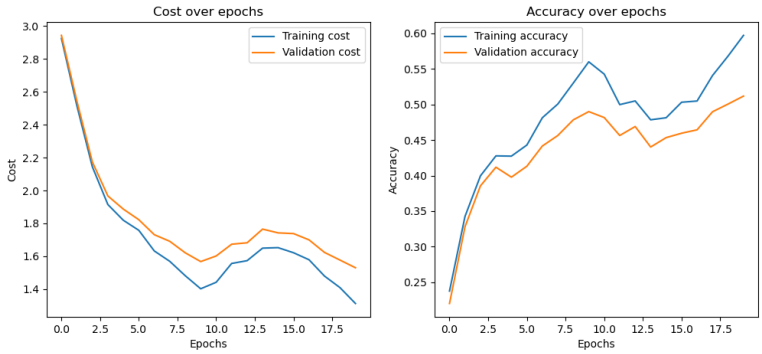

Figure 6: Cost function and accuracies with BN for 9-layer NN. Test accuracy was 52.20%

reg_factor	1e-5	1e-4	1e-3	1e-2	1e-1
val_acc in %	51.00	51.44	52.14	51.38	24.30

Table 1: First grid search for the optimal regularization factor

reg_factor	5e-4	1e-3	5e-3	1e-2	5e-2
val_acc in %	51.34	53.18	53.62	53.44	52.38

Table 2: Second grid search for the optimal regularization factor

2.3 Grid search for lambda

To optimize the hyper parameter λ I chose to test it from $\lambda = 5e{-}4$ to $\lambda = 5e{-}2$. The results can be seen in Table 1 and 2. In general the regularization factor does not seem to have a strong influence on the networks performance and running the experiment with the same λ twice resulted in up to 1% validation accuracy differences. Coincidentally the default value of 0.005 performed best which is why I kept $\lambda = 0.005$ for the remainder of this assignment.

2.4 Sensitivity to initialization

I adapted my code to take `sig` as an input variable for the variance of the weight initialization and tested `sig = 1e-1`, `1e-3`, and `1e-4`. In total I ran 6 experiments with a 3-layer network, each variance with and without BN. The results are shown in Table 3.

Without BN the network diverged resulting in 10% accuracy (random guessing). This might be due to the high learning rates used in the cyclic learning rate scheme. BN seems to stabilize the training process.

	`sig=1e-1`	`sig=1e-3`	`sig=1e-4`
without BN	9.98	10.05	10.14
with BN	53.83	53.48	54.14

Table 3: Validation accuracies for different initialization variances; with and without Batch Normalization

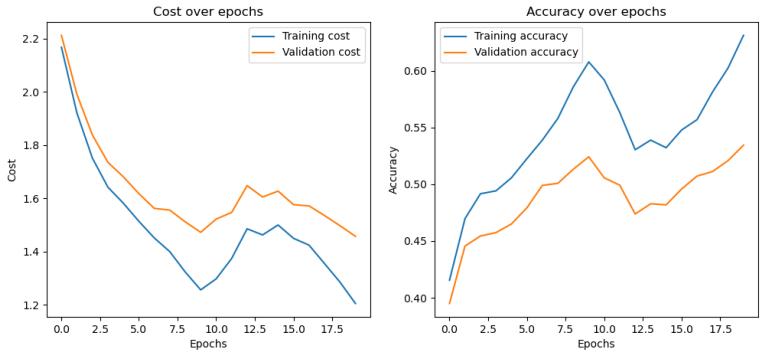

Figure 7: Cost function and accuracies with BN after activation layer for 3-layer NN. Test accuracy was 53.71%

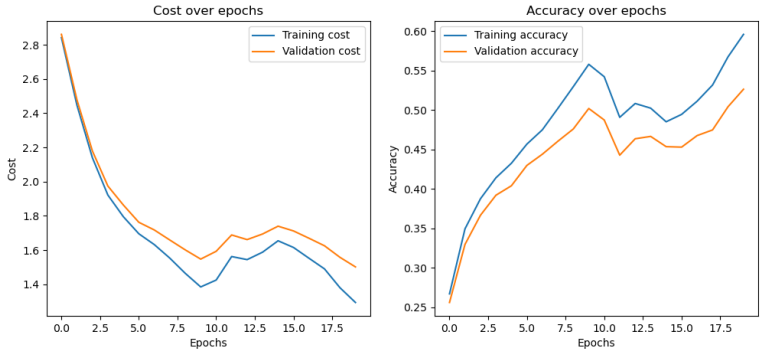

Figure 8: Cost function and accuracies with BN after activation layer for 9-layer NN. Test accuracy was 52.27%

3 Bonus assignment

3.1 Explore Batch Norm some more

BN after activation layer

Due to my modularity of my code this task was as easy as changing the order of the layers. As seen in Figure 7 for the 3-layer network the change in test accuracy from 53.89% to 53.71% is not significant. For the 9-layer network the test accuracy changed from 52.20% to 52.27 % shown in Figure 8.

Mean and variance batch computation

To accommodate the different ways to calculate the mean and variance during training and test time, I added an additional parameter `aggregation_mode` to my BN layers. The default setting is `'ema'` for Exponential Moving Average

(EMA). The two new settings are `'pbn'` for PreciseBN and `'abn'` for Adaptive BN. I limited my test to run for the 9-layer network as I think this is the more interesting network to study as BN gets increasingly important the deeper the networks becomes.

For the PreciseBN I aggregate the sample means and variances during training compute the averages of those samples every 10 mini batches. For a mini batch size of 100 this means the running mean and variance is computed once per 1000 images. In my experiments this does not seem to have any significant effect on the final test accuracy as it went from 52.20% with EMA to 51.77% with PreciseBN.

For Adaptive BN the mean and variance is computed during test time on the test set itself. However, this did also not yield changes to the networks performance. Without augmentation of the test data the accuracy was 52.19%, again no change to EMA. If the test data is augmented in a similar fashion as in assignment 2 (translation and flipping) the accuracy drops: 10% data augmentation resulted in 49.56% test accuracy, 50% augmentation yielded 38.52% and if 100% of the images are augmented the test accuracy dropped to 28.97%.

In summary, I could not achieve any performance increases by changing the order of activation and BN layer nor by using different ways to compute the mean and variance. Thus, I will leave the BN layer in front of the activation layer for the remainder of the assignment as well as continue using EMA in my BN layers.

3.2 Can you make your deep network easier to optimize

Adam Optimizer

I added the Adam Optimizer according to the pseudo code implementation in Figure 9. I did some limited hyperparameter search and reached a final test accuracy of 52.76% with a cyclic learning rate with `eta_min = 1e-5`, `eta_max = 1e-3`, `n_s = 2250` for 8 cycles and a slight learning rate decay with a 7-layer network with hidden nodes [60, 50, 40, 30, 20, 10]. The results of the training can be seen in Figure 10.

Increase number of hidden nodes

As Adam did not yield any better results for this experiment I reverted back to the basic weight update method with cyclic learning rates. For a 7-layer network I tried two hidden layer configurations [256, 128, 64, 48, 32, 16] and [512, 256, 128, 64, 32, 16]. These networks achieved 57.15% and 58.20% respectively. These results lead to the conclusion that deeper networks are capable of learning better generalization for the trade-off of long training times of more than 30 minutes on my machine. The training process can be seen in Figure 11 and 12.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 9: Pseudo code implementation of Adam Optimizer

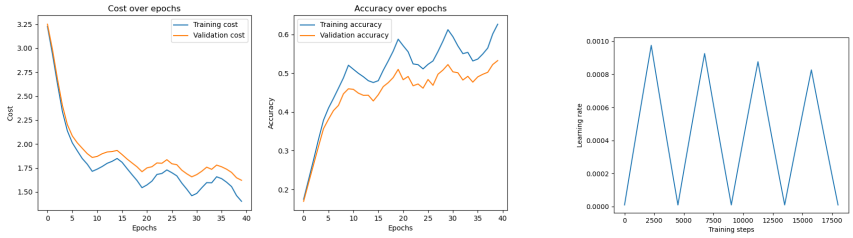

Figure 10: Performance with Adam optimizer and cyclic learning rate. Final test accuracy 52.76%

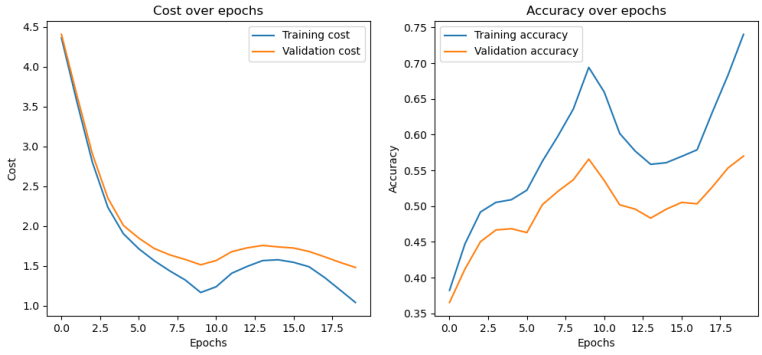

Figure 11: 7-layer network with hidden layers [256, 128, 64, 48, 32, 16]. Test accuracy was 57.15%

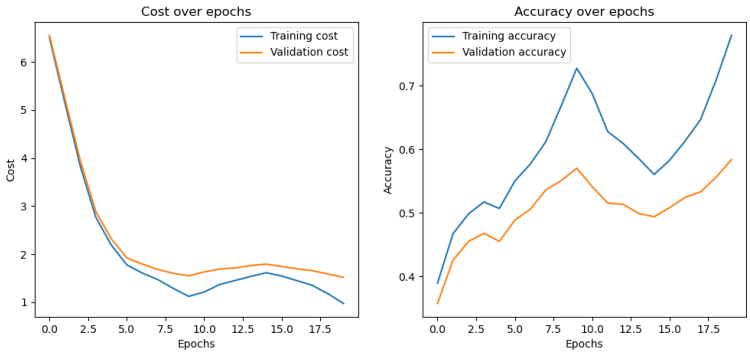

Figure 12: 7-layer network with hidden layers [512, 256, 128, 64, 32, 16]. Test accuracy was 58.20%