DD2424 - Assignment 3

Summary

I have implemented `batch normalization` and trained a k-layer network using cyclical learning rates. For the 3-layer network with `batch normalization` and 50 nodes in each hidden layer, I have implemented a λ-search and then trained the top-performing parameter setting for 3 cycles, for which I achieved a 53.60% accuracy on the test set. However, this was still lower than for the standard $\lambda = 0.005$, for which accuracy on the test set was 53.76% with `batch normalization`. For the 9-layer network, the corresponding performance with and without `batch normalization` was received 50.83% and 46.35%, respectively. However, as is clarified in the report, I have not achieved a perfect match for the analytical and numerical gradient calculations. More on this below.

The code for the assignment has been written in `python`. I have implemented the neural network as a class. For the hand-in, all of the code has been put toghether in a main file with all the functions and the class declared at the top. For the hand-in, I have also commented out the saving of generated figures and results in JSON files, as well as omitting some of the case-specific testing and gradient testing.

Oskar Stigland

DD2424
Spring 2023

Checking gradients

In order to validate the anaytic gradient computations, I compared the analytically computed gradients with numerically computed gradients on a subset of the weights of a large subset of the training data using $h = 10^{-5}$. I got that

$$||\nabla_{\boldsymbol{w}} \boldsymbol{J} - \nabla_{\boldsymbol{w}} \boldsymbol{J}^{\text{numerical}}||_{\infty} < 10^{-9}, \quad \forall \boldsymbol{w} \in \boldsymbol{W}_k$$

where $\boldsymbol{W}_k$ is the full set of weights for the final layer, including also the weights in the bias vector. I also had that

$$||\nabla_{\boldsymbol{w}} \boldsymbol{J} - \nabla_{\boldsymbol{w}} \boldsymbol{J}^{\text{numerical}}||_{\infty} < 10^{-9}, \quad \forall \boldsymbol{w} \in \boldsymbol{\gamma}_{k-1}, \boldsymbol{\beta}_{k-1}$$

where $\boldsymbol{\gamma}_{k-1}, \boldsymbol{\beta}_{k-1}$ are the full set of weight in the scale, γ_{k-1}, and shift, β_{k-1}, for the next-to-last layer, i.e. $\ell = k - 1$. However, for the remaining weights in the network the difference was on the order of 10^{-2}. The issue seems to arise when the gradients are propagated through the batch normalization, i.e. refered to as the `batchNormBackPass`. I have turned my code inside out several times, but can not seem to find the issue. It might also be that there is an issue in the numerical gradient calculations. When training a 5-layer network with $[50, 50, 20, 30]$ hidden nodes - as well as a 9-layer network with the same specs as in the assignment description - on a small subset of the training data with $|D| = 100$ and 500 epochs, not using any regularization, I get the following results:

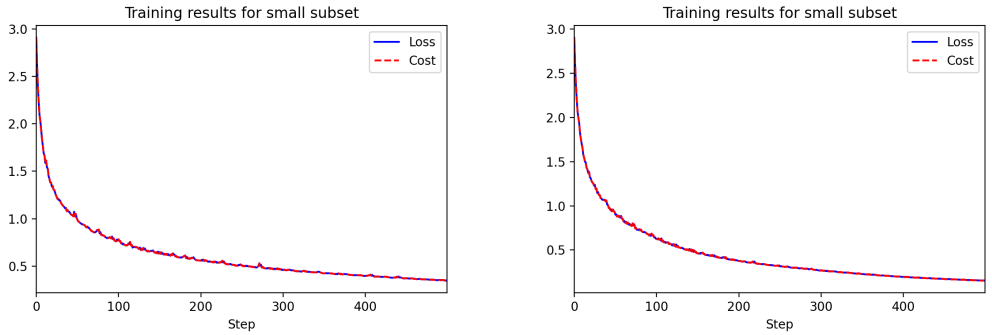

Figure 1: Loss and cost for 5-layer and 9-layer network using `batch normalization` and $\eta = 0.01$, $\lambda = 0.0$, on $D \subset \boldsymbol{X}$, $|D| = 100$

Thus, I am confident the analytically computed gradients are **not** entirely bug free. Especially, the loss and cost gets increasingly unstable as the number of layers increase. However, despite the analytical gradients not being entirely bug-free, the multi-layer network seems to learn how to classify the CIFAR-10 fairly well, with testing accuracy above 50% during a normal training procedure with batch normalization and a cyclical learning rate.

For the k-layer network **without** `batch normalization`, the analytical gradients are seemingly bug free, with pairwise absolute difference between analytically and numerically computed gradients on the order of 10^{-10} for all weights.

The effects of `batch normalization`

Training a 3-layer network

For the 3-layer network, there is no discernable difference in terms of the performance of the networks with respect to the accuracy on the test dataset. In fact, omitting `batch normalization` seems to result in a more stable training procedure.

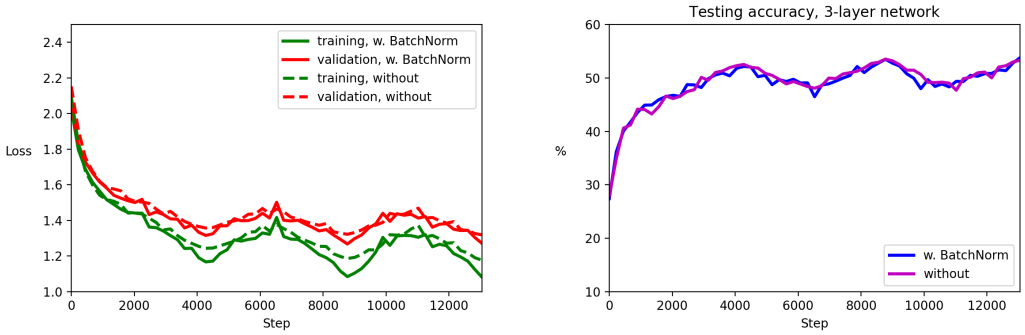

Figure 2: Loss and accuracy for 3-layer network with and without `batch normalization`, using $\lambda = 0.005$, and a cyclical learning rate with $\eta_{min} = 10^{-5}$ and $\eta_{max} = 10^{-1}$.

Training a 9-layer network

There is a clear difference when implementing and omitting `batch normalization` for the 9-layer network. In the former case, the network achieves an accuracy on the test dataset slightly above 50%, while the corresponding accuruacy in the latter case is on the order of 45%. There is also a clear difference in terms of the loss, where the network trained with `batch normalization` achieves a lower loss across the training process. That is, the network with `batch normalization` achieves better performance and learns significantly faster compared to the network without it.

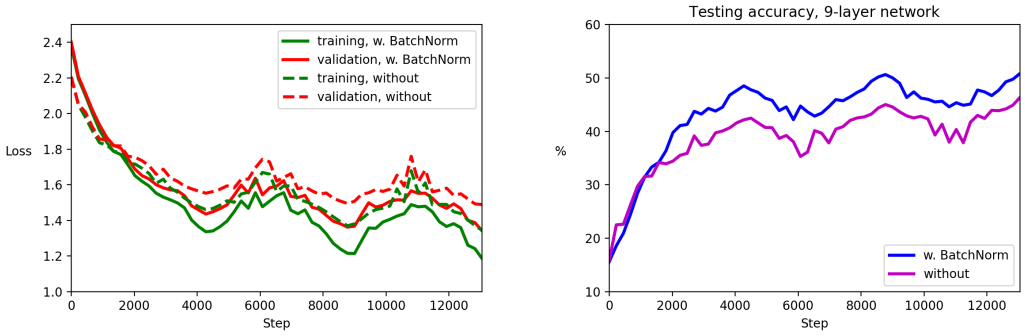

Figure 3: Loss and accuracy for 9-layer network with and without `batch normalization`, using $\lambda = 0.005$, and a cyclical learning rate with $\eta_{min} = 10^{-5}$ and $\eta_{max} = 10^{-1}$.

Conducting a λ-search

I conducted a search for an optimal λ by computing 10^l where

$$l = l_{min} + (l_{max} - l_{min}) \cdot u, \quad u \sim \text{Uniform}(0, 1)$$

and $l_{min} = -4$ and $l_{max} = -2$, by sampling u for a number of iterations and training for 2 cycles per iteration. I achieved the best highest accuracy on the test dataset with $\lambda = 0.0093$. Then, I trained a network with $\lambda = 0.009$ for 3 cycles and attained an accuracy of 53.60%. The loss, cost and accuracy are plotted below. In general, it seems that the multi-layer network with `batch normalization` benefits from a higher rate of regularization in comparison to the network in the previous assignment, where I found an optimal λ to be more on the order of ≤ 0.0002. This might, of course, be primarily due to the inclusion of more layers, rather than the effect of `batch normalization`. And while 53.60% is a fair result for a 3-layer network, it should be noted that this was the top-performing network for the particular λ-search that I conducted, and that a network with $\lambda = 0.005$ actually performed slightly better in terms of accuracy on the test data.

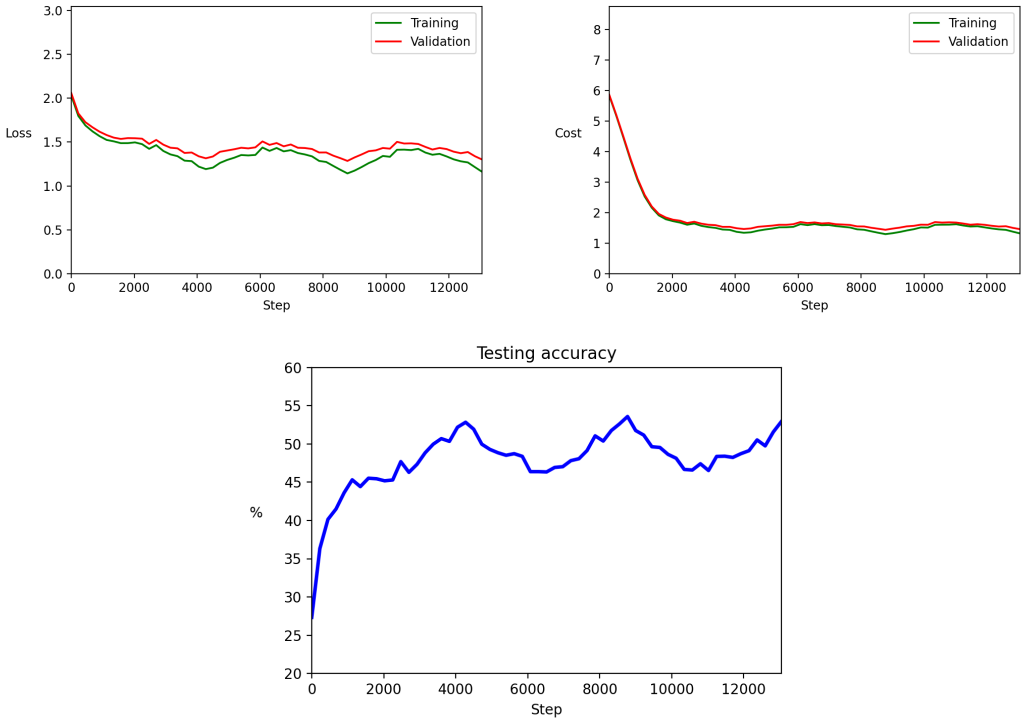

Figure 4: Loss , cost and testing accuracy for 3-layer neural network with batch normalization and $m = 50$ nodes in each hidden layer, using $\lambda = 0.009$, $\eta_{min} = 10^{-5}$, $\eta_{max} = 10^{-1}$, and $ns = 2250$ for 3 cycles.

Sensitivity to initialization

When we decrease the variance in the random normal initialization of $\boldsymbol{W}$, we get an increasingly poor training performance in the case where `batch normalization` is omitted. In the second case, where $\boldsymbol{W}$ are initialized with $\mathcal{N}(0, 0.001)$, the network seems to start learning first after some $4000 - 5000$ steps. Eventually, the achieved accuracy is very similar to that of the network with `batch normalization`. However, in the third case, where $\boldsymbol{W}$ are initialized with $\mathcal{N}(0, 0.0001)$, there is in fact no learning whatsoever, even after the full 3 cycles. For the corresponding network with `batch normalization`, the loss during training and accuracy during testing seems to be pretty much equal as that of the two previous cases. Hence, this clearly demonstrates the value of using `batch normalization` - as well as the importance of appropriate weight initialization.

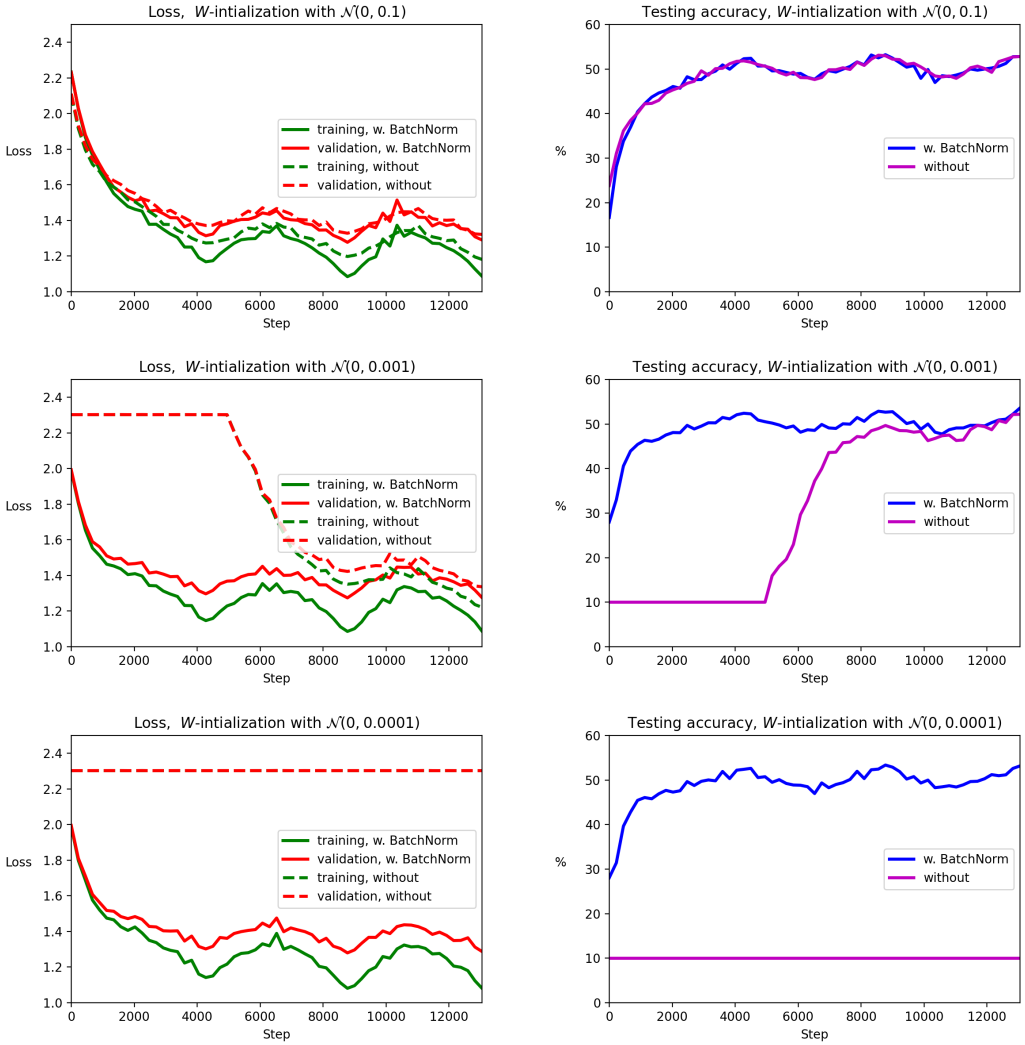

Figure 5: Loss and accuracy for 3-layer network with and without `batch normalization`, using $\lambda = 0.005$, and a cyclical learning rate with $\eta_{min} = 10^{-5}$ and $\eta_{max} = 10^{-1}$, as well as random normal initialization for $\boldsymbol{W}$, with $\sigma \in \{10^{-1}, 10^{-3}, 10^{-4}\}$.