Assignment 3 Report

DD2424, Kristian Fatohi

1. Introduction

The objective of Lab Assignment 3 is to develop and train a k-layered neural network designed to classify the diverse categories present in the CIFAR-10 dataset. This network will utilize batch normalization and the optimization of its parameters will be achieved through the implementation of mini-batch gradient descent.

2. Analytic and Numerical gradient computations

To check whether the analytical gradient computations are valid, I compared them to numerical gradients since they are practically assumed to be very accurate but computationally expensive. Due to time constraints, I have used a reduced dimensionality of the samples and small batches to compute in a reasonable amount of time. To maintain consistency throughout this small comparison, the initial weights and biases were systematically varied using seeds. These seeds are 10 and 100. The results are shown in the tables below where each experiment employs 100 features:

(W_1, W_2, W_3)	$1.384 * 10^{-11}$	$1.305 * 10^{-11}$	$1.411 * 10^{-11}$
(B_1, B_2, B_3)	$2.194 * 10^{-12}$	$2.894 * 10^{-18}$	$1.222 * 10^{-11}$
(γ_1, γ_2)	$1.523 * 10^{-11}$	$1.379 * 10^{-11}$	-
(β_1, β_2)	$1.338 * 10^{-11}$	$1.477 * 10^{-11}$	-

Table 1: Comparison between numerical and analytical computations of gradients over the first 1000 samples using the seed 10.

(W_1, W_2, W_3)	$1.426 * 10^{-11}$	$1.402 * 10^{-11}$	$1.391 * 10^{-11}$
(B_1, B_2, B_3)	$3.622 * 10^{-12}$	$4.532 * 10^{-13}$	$1.456 * 10^{-11}$
(γ_1, γ_2)	$1.351 * 10^{-11}$	$1.301 * 10^{-11}$	-
(β_1, β_2)	$1.498 * 10^{-11}$	$1.301 * 10^{-11}$	-

Table 2: Comparison between numerical and analytical computations of gradients over 1000 random samples using the seed 10.

(W_1, W_2, W_3)	$7.024 * 10^{-8}$	$8.782 * 10^{-8}$	$1.325 * 10^{-10}$
(B_1, B_2, B_3)	$1.342 * 10^{-12}$	$2.771 * 10^{-18}$	$1.370 * 10^{-11}$
(γ_1, γ_2)	$1.500 * 10^{-11}$	$1.455 * 10^{-11}$	-
(β_1, β_2)	$1.634 * 10^{-6}$	$2.260 * 10^{-7}$	-

Table 3: Comparison between numerical and analytical computations of gradients over the first 1000 samples using the seed 100.

(W_1, W_2, W_3)	$3.257 * 10^{-6}$	$2.095 * 10^{-8}$	$1.332 * 10^{-11}$
(B_1, B_2, B_3)	$9.004 * 10^{-13}$	$3.283 * 10^{-18}$	$1.619 * 10^{-11}$
(γ_1, γ_2)	$1.563 * 10^{-11}$	$1.381 * 10^{-11}$	-
(β_1, β_2)	$1.128 * 10^{-6}$	$8.847 * 10^{-10}$	-

Table 4: Comparison between numerical and analytical computations of gradients over 1000 random samples using the seed 100.

Studying these results shown in all four tables, we can see that the difference between the gradients calculated analytically and those computed numerically are quite small, with the largest being on the order of 10^{-6}. This suggests that the gradient computations were somewhat bug-free. Why the values reflect different values depending on what samples are used, is because of the back-propagation of gradients.

3. 3-layer network

Next, I test the effect of batch normalization on a 3-layer network. The network is composed as instructed, of 50 neurons in the hidden layers and ten neurons in the output layer. The network is also initialized with **He initialization.** The table below shows the hyperparameter used for training of the network.

λ	η_{min}	η_{max}	η_s	Cycle
0.005	$1 * 10^{-5}$	*0.1*	*5*45000/*η_{batch}	*2*

Table 5: Hyperparameters for the 3-layer network.

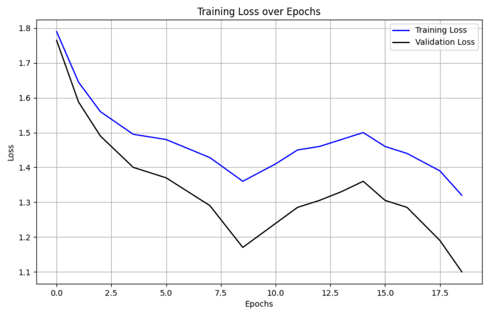

Figure 1: **With** batch normalization.

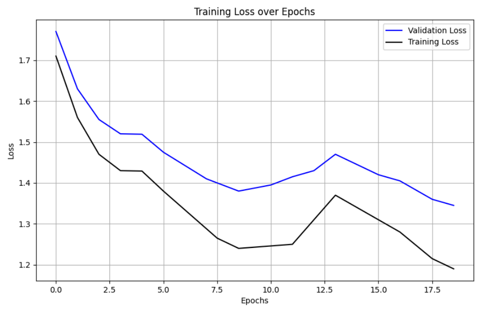

Figure 2: **Without** batch normalization.

By just examining the two plots, you can see with your own eye that the one with batch normalization is a tad smoother but other than that, there really is not a huge difference. This could also be because three layers are not that "deep" enough when training the network. However, besides the smoothness, we can also see that Figure 1, manages to achieve a lower loss value than Figure 2, including batch normalization edges the option to leave it out but further experimenting is needed to aid that statement further.

4. 9-layer network

As stated earlier, further experimenting is needed, and now I will experiment with a 9-layer network. To just examine the factor of the number of layers, the hyperparameters stay the same with the **He initialization,** the specific hyperparameters can be shown in Table 5, the only difference is the architecture of the network, which is [50, 30, 20, 20, 10, 10, 10, 10].

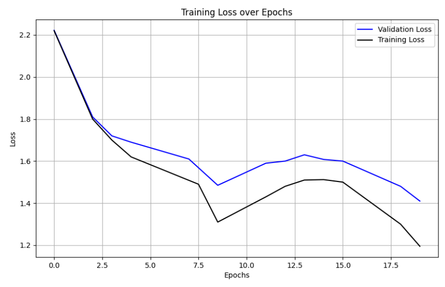

Figure 3: **With** batch normalization.

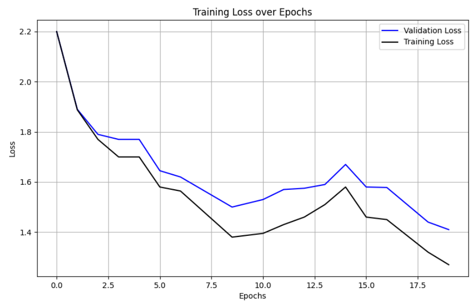

Figure 4: **Without** batch normalization.

The figures above really show us how batch normalization has an effect. We still see the smoothness as with the plots from the 3-layer network however, here they are more clear to see and we also manage to achieve a lower loss value with batch normalization. I can with confidence say that batch normalization is a good thing!

5. Lambda search

To further examine if we can achieve a better accuracy rate, lambda search is the next experiment. The search is done with the range of lambda values being tested as follows:

$$10^{-x}, \; where\ x\ is\ \{5,\ 4.5,\ 4,\ 3.5,\ 3,\ 2.5,\ 2,\ 1.5,\ 1\}$$

With these, we search until we get 10 values from the coarse search where the two best-performing values in terms of accuracy are used for the fine search. The coarse search is done on a 3-layer neural network with a batch size of 100, step size of 2250, and 2 cycles of learning. The table below displays the top three most accurate lambdas:

Lambda	10^{-2}	$10^{-2.5}$	10^{-3}
Accuracy	0.5395	0.5385	0.5380

Table 6: Accuracy for the top three lambdas.

Using the results from the coarse search we can proceed with the fine search to improve the accuracies. The two that provided the best accuracies are further used to and the distance between those two values is used to define a new interval. This interval is defined as follows:

$$[best_{\lambda} - dist,\ best_{\lambda} + dist]$$

Utilizing this interval it is possible to look for values around the best ones provided by the previous search. The table below displayed the top three most accurate lambdas for the fine search:

Lambda	$\sim 10^{-2.32}$	$\sim 10^{-2.30}$	$\sim 10^{-2.03}$
Accuracy	0.5407	0.5397	0.5395

Table 7: Accuracy for the top three lambdas.

The fine search was also done with 2 cycles of learning and using the best lambda value from the fine search, $\sim 10^{-2.32}$, I trained the network and achieved an accuracy of 53.94%, now that puts a smile on my face!

6. Sensitivity to initialization

This part is an experiment on the initializing aspect. Instead of using **He initialization,** I initialize the weights of the neural network. The hyperparameters of the network are the same as previously mentioned in Table 5. The initialization of the weights will be normally distributed with sigmas. Three experiments will be done where the sigma value and as before, batch normalization will be alternately turned off and on for each experiment.

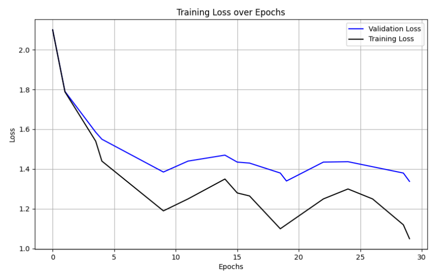

Figure 5: With batch normalization and σ=0.1.

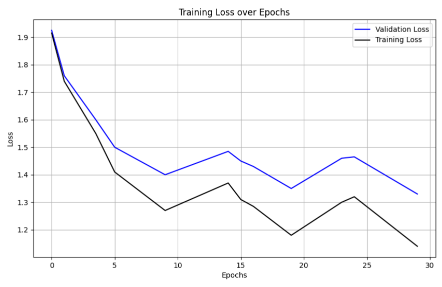

Figure 6: Without batch normalization and σ=0.1.

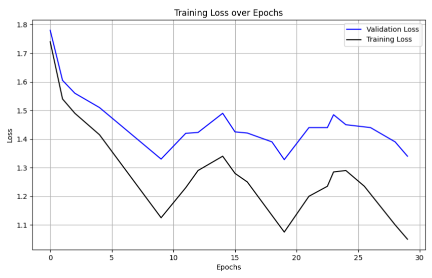

Figure 7: Without batch normalization and σ=1e-3.

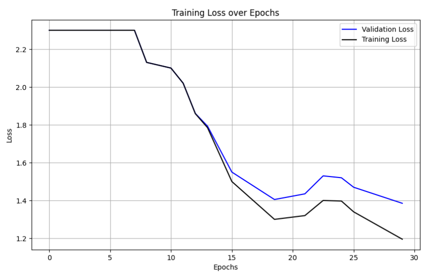

Figure 8: Without batch normalization and σ=1e-3.

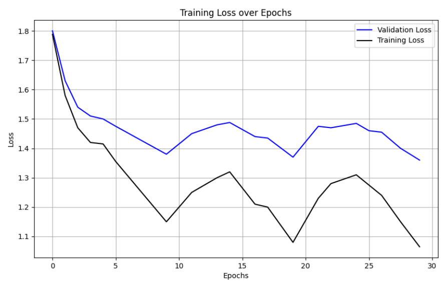

Figure 9: Without batch normalization and σ=1e-4.

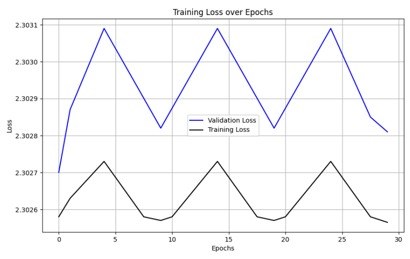

Figure 10: Without batch normalization and σ=1e-4

There is a lot to learn from the figures above, we can see clearly that loss is greater for the experiments without batch normalization but when the sigma gets smaller, the loss becomes greater if we exclude batch normalization. This implies that batch normalization can remove a large part of the dependency on the weight initialization. In the last test, where the sigma is the smallest, the network can not really train since the weights are close to each other. However, the easiest conclusion one can make is that batch normalization can save the day on a rainy day with sigma values.