Assignment 2 Report

DD2424, Kristian Fatohi

1. Introduction

The objective of Lab Assignment 2 is to develop and evaluate a two-layer neural network designed to classify the diverse categories present in the CIFAR-10 dataset. This network will utilize a cross-entropy loss function combined with L_2 regularization, and the optimization of its parameters will be achieved through the implementation of mini-batch gradient descent.

2. Analytic gradient computations

To check whether the analytical gradients computations are valid, I compared them to numerical gradients since they are practically assumed to be very accurate but computationally expensive. Analytical gradients are computed using explicit formulas while numerical gradients are calculated using finite difference methods. To save time during the computation, I have decided to use small batches from the CIFAR-10 dataset and only do two tests. To maintain consistency throughout this small comparison, the initial weights and biases were systematically varied using seeds. These seeds are 10 and 100.

Gradient	**Numerical**	**Analytical**
(W_1, W_2)	$(4.312 * 10^{-8}, 1.923 * 10^{-7})$	$(4.535 * 10^{-14}, 3.895 * 10^{-13})$
B_1, B_2	$(4.310 * 10^{-8}, 3.988 * 10^{-7})$	$(7.612 * 10^{-13}, 4.443 * 10^{-18})$

Table 1: Comparison between numerical and analytical computations of gradients over the first 20 samples using the seed 10.

Gradient	**Numerical**	**Analytical**
(W_1, W_2)	$(4.119 * 10^{-8}, 1.812 * 10^{-7})$	$(2.266 * 10^{-14}, 9.871 * 10^{-13})$
B_1, B_2	$(4.120 * 10^{-8}, 4.094 * 10^{-7})$	$(3.448 * 10^{-12}, 4.123 * 10^{-12})$

Table 2: Comparison between numerical and analytical computations of gradients over the first 20 samples using the seed 100.

Studying these results shown in both Table 1 and 2, 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^{-7}. This suggests that the gradient computations were somewhat bug-free.

3. Cyclical Learning Rates

Exploring how the CLR or cyclical learning rates allow our classifier to optimize, I did two models, each with different configurations, and provided figures for the cost/loss and accuracy for each configuration. CLR works as its name intends, it cycles the learning rate during training, and it transitions between a large and a small value. During the whole training period, this cycling process is repeated.

3.1 Model 1

Hyperparameters:

Total Epoch	Batch Size	λ	η_{min}	η_{max}	Step Size	Cycle
10	*100*	*0.01*	$1 * 10^{-5}$	*0.1*	*500*	*1*

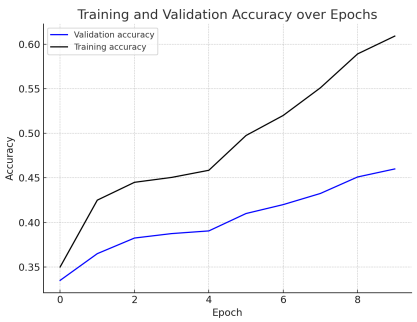

Figure 1: Training and Validation accuracy over Epochs

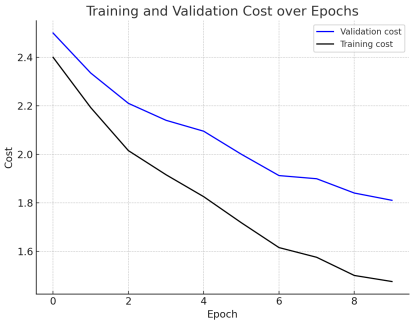

Figure 2: Training and Validation cost over Epochs

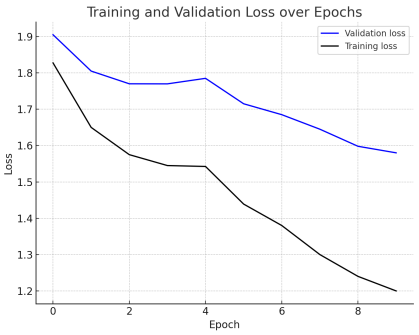

Figure 3: Training and Validation loss over Epochs

3.2 Model 2

Hyperparameters:

Total Epoch	Batch Size	λ	η_{min}	η_{max}	Step Size	Cycle
48	*100*	*0.01*	$1 * 10^{-5}$	*0.1*	*800*	*3*

Figure 4: Training and Validation loss over Epochs

Figure 5: Training and Validation cost over Epochs

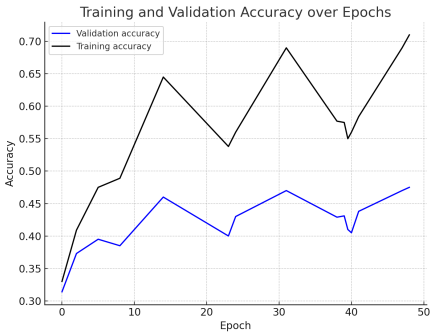

Figure 6: Training and Validation accuracy over Epochs

3.3 Discussion

By viewing all figures for both models, we can see that cyclical learning rates as a method is characterized by a triangular wave. If you think of the concept behind CLR then the triangular shapes are really not so unexpected, when the learning rate rises and shrinks it would obviously give a kind of zigzag visualization. Analyzing the accuracy test for each model, we can also see that model 2 is slightly more accurate. Model 1 clocked in at around 44.89% while model 2 got a 46.12%. This, however, is likely due to the increased step size and the amount of cycles performed.

4. Coarse Search

Exploring the use of coarse search will hopefully help find the optimal regularization parameter since the two models above seem to overfit. To accomplish this the range of lambda values being tested are as follows:

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

The hyperparameters used are:

Batch Size	η_{min}	η_{max}	**Step Size**	**Cycle**
100	$1 * 10^{-5}$	*0.1*	*900*	*2*

using these parameters, these are the following regularization parameter that received the best accuracy over the validation set:

Lambda	$10^{-2.5}$	$10^{-4.5}$	10^{-3}
Accuracy	0.5205	0.5184	0.5179

5. Fine Search

Using the results from the coarse search we can proceed with the fine search to improve the accuracies. In fine search, we aim for a more precise search for the lambda parameter so we modify our range of lambda values to the following:

$$10^{-x},\ where\ x\ is\ \{4.5,\ 4,\ 3.5,\ 3,\ 2.5\}$$

Now for fine search to do its magic, I generated random lambda values within the range mentioned above so that it can produce a more specific value. I use four cycles this time instead of two for a more exhaustive search. Using this, I managed to obtain the following;

Lambda	$\sim 10^{-2.94}$	$\sim 10^{-2.89}$	$\sim 10^{-2.91}$
Accuracy	0.5386	0.5251	0.5244

Examining the improvement, we can conclude that fine search produces better accuracies than the coarse search.

6. Lambda Setting

Finally, at the last step, after finding a near-optimal lambda value, we can train the network once again, this time, I use almost the entire dataset and the last 1000 samples for validation. I managed to achieve a 52.34% accuracy over the test data and these are the following figures for the tests done.

Figure 7: Cost over training Epochs

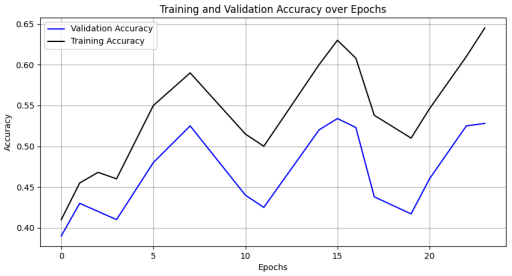

Figure 8: Accuracy over training Epochs

Figure 9: Loss over training Epochs

As the figures display, having a good lambda value and increasing the amount of data used does improve the generalization of the model and can produce better accuracies.