CS5260 Assignment 6 – Colossalai and LR Range Test

Gin Wen Ng
A0142949U
Department of Computer Science
National University of Singapore
ginwenng@u.nus.edu

Assignment Overview

The objective of this assignment is to study the use of Learning Rate Range by using the Colossai framework. The assigned model and dataset to use for this study are the Lenet5 model and the MNIST dataset respectively. The task is to choose one optimizer and two different learning rate scheduling methods and write a brief report about the corresponding LR region on LR range test plot.

1 Introduction

The code used for this assignment can be found in https://github.com/ginwenng/CS5260_Assignment_6. The key tasks done for the assignment can be found in Section 4. It provides information on the experiment outputs and observations as a result of trying out different optimizer algorithms and learning rate scheduling methods.

Sections 2 and 3 mainly covers an explanation of my understanding of the different optimizer algorithms and the learning rate scheduling methods. It also gives information on the sources of implementations of the algorithms in my experiments.

2 Optimizer

When training neural networks, we use optimizers to modify the weights and learning rate of the neural network given the gradients. This is to help reduce the losses. There are a number of optimizers available such as SGD, Adam, LARS and LAMB.

SGD stands for Stochastic Gradient Descent. It is an efficient and effective optimization method that has become one of the most commonly used and most foundational algorithms for both traditional machine learning or deep learning. (Kingma and Ba, 2014) However, as SGD's algorithm involves iteratively updating parameters by moving in the direction of the gradient, it is sequential and hence it does not scale as well. (You et al., 2019) The implementation of the SGD Optimizer can be taken from the PyTorch library[1].

Adam requires only first-order gradients and hence, it has little memory requirement. In addition, it can handle sparse gradients and both on-line and non-stationary settings as it has managed to leverage on popular methods such as AdaGrad and RMSProp. (Kingma and Ba, 2014) However, it is also often debated that models trained with the Adam optimizer does not generalize well. The implementation of the Adam Optimizer can be taken from the PyTorch library[2].

LARS stands for Layerwise Adaptive Rate Scaling. The authors of the paper has explained that LARS uses a separate learning rate for each layer instead of each weight, which would help improve

[1] https://pytorch.org/docs/stable/generated/torch.optim.SGD.html
[2] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam

Preprint. Under review.

stability. Also, there is better control of training speed as the magnitude of the update takes the weight norm as reference. (You et al., 2017) The implementation of LARS is taken from an Open Source Github Repository[3].

LAMB is a layerwise adaptive large batch optimisation technique. LAMB is inspired by LARS where layerwise adaptive learning rates was used. (You et al., 2019) The implementation of the LAMB Optimizer can be taken from the pytorch-lamb library. Although the official implementation is available on Tensorflow[4], the PyTorch implementation of LAMB is taken from an Open Source Github Repository[5].

3 Learning Rate Scheduling Method

When training neural networks, we use learning rate schedulers to modify the learning rate. This is to help improve the learning. There are a number of learning rate scheduling methods available such as exponentially increasing the learning rate from low to high, Multistep and OneCycle.

For the first method, the learning rate would start low and gradually increase exponentially from low to high. The implementation can be taken from the PyTorch library[6] and parsing the exponential increase function as a parameter.

Multistep would decay the learning rate of each parameter group by gamma once the number of epoch reaches one of the milestones. The implementation can be taken from the PyTorch library[7].

Onecycle would set the learning rate of each parameter group according to the 1cycle learning rate policy. The 1cycle policy anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate. (Smith and Topin, 2017) The implementation can be taken from the PyTorch library[8].

4 Experiments

As mentioned previously, the assigned model and dataset to use for this study are the Lenet5 model and the MNIST dataset respectively.

There would be multiple experiments conducted as four different Optimizers would be tested, two of which there are no readily-usable function from PyTorch. Three different LR scheduling method would also be used. The batch size and the number of epochs used to train the neural network would be 128 and 30 respectively.

[3]https://github.com/JosephChenHub/pytorch-lars/blob/master/optimizers/lars.py
[4]https://github.com/tensorflow/addons/blob/master/tensorflow_addons/optimizers/lamb.py
[5]https://github.com/cybertronai/pytorch-lamb
[6]https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.LambdaLR.html
[7]https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.MultiStepLR.html
[8]https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html

4.1 Optimizers

First, I wanted to explore the different optimizers. During this experiment, the LR Scheduling Method used is to exponentially increasing the learning rate from low to high, and parameters such as the learning rate and the weight decay are set as 0.1 and 0.0005 respectively.

4.1.1 Stochastic Gradient Descent

In Figure 5, we can see that while the learning rate exponentially increases, the loss reduces during training. For the test dataset, we can also see the loss reducing.

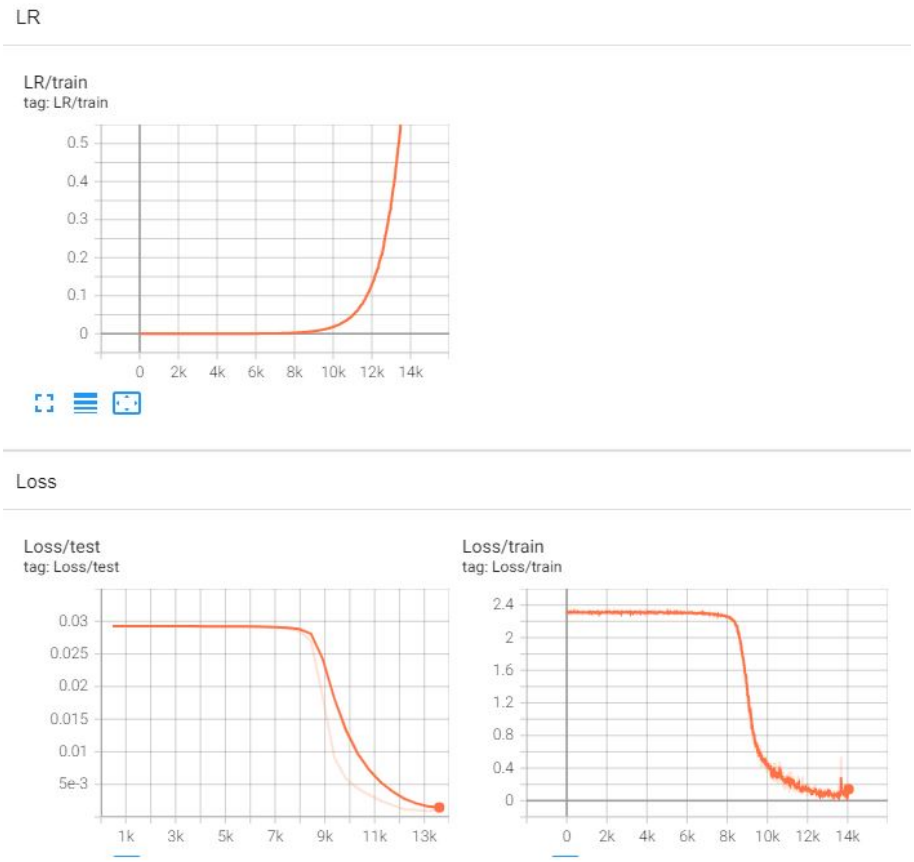

Figure 1: Stochastic Gradient Descent

4.1.2 Adam

In Figure 2, we can see that while the learning rate exponentially increases, the loss start off small but reduces only slightly during training, followed by an unstable increase and reduction, with an overall increase in loss. For the test dataset, we can also see a similar pattern where the loss reduces slightly and increases exponentially later on.

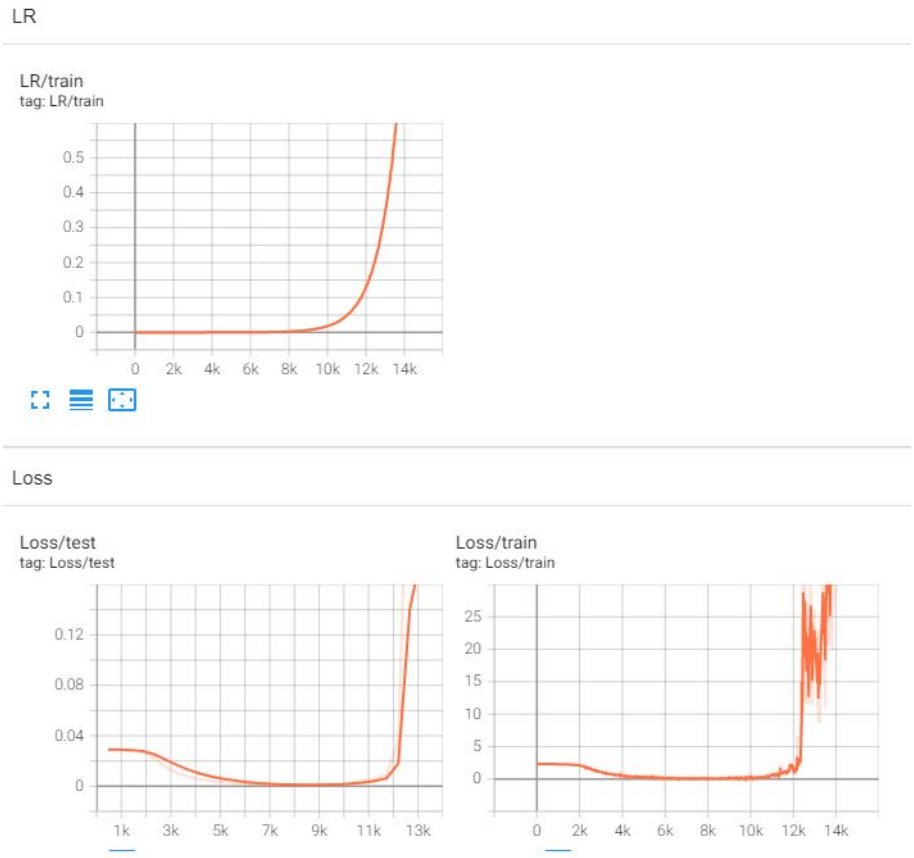

Figure 2: Adam

4.1.3 LARS

In Figure 3, we can see that while the learning rate exponentially increases, the loss start off small but does not reduce during training. This is followed by an expontial increase.. For the test dataset, we can also see a similar pattern where the loss starts small and increases exponentially later on.

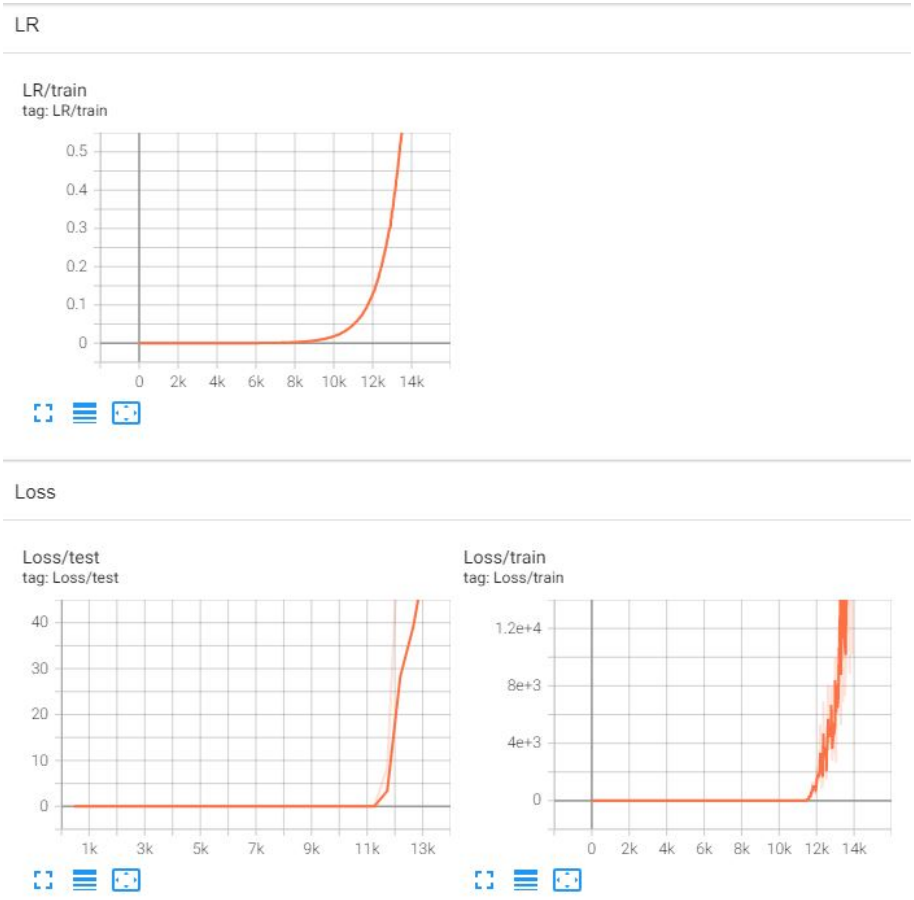

Figure 3: LARS

4.1.4 LAMB

In Figure 4, we can see that while the learning rate exponentially increases, the loss start off small but reduces only slightly during training, followed by increasing exponentially. For the test dataset, we can also see a similar pattern where the loss reduces slightly and increases exponentially later on.

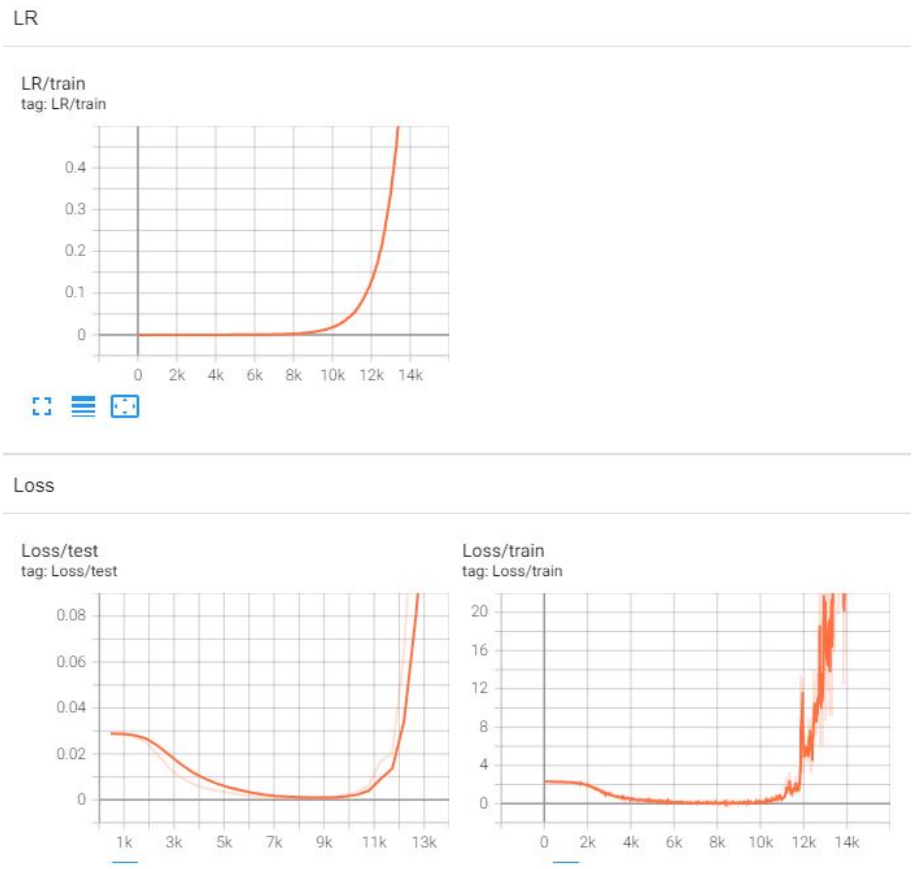

Figure 4: LAMB

4.1.5 Conclusion

The SGD Optimizer, given the same set of parameters and conditions, is able to help reduce loss unlike the other optimizers tried. It can however be further improved as we can note from the diagram that there is some unstability towards the end. More enhancement are done in the next section where the learning rate is being modified using various learning rate scheduling methods.

4.2 Learning Rate Scheduling Methods

Next, I wanted to explore the different learning rate scheduling methods. During this experiment, the Optimizer used is SGD, and parameters such as the learning rate and the weight decay are set as 0.1 and 0.0005 respectively.

4.2.1 Exponentially Increasing Learning Rate from Low to High

In Figure 5, we can see that while the learning rate exponentially increases, the loss reduces during training. For the test dataset, we can also see the loss reducing. We can see that although the loss is reducing in general during the training process, the curve is quite flat at the start, and towards the end it is not smooth and there is some wavering. This means that the learning rate in the region where 0 to 8k data is being trained is too small and that the learning rate in the region where 12k to 14k data is being trained is getting too large. As such, we can observe from the learning rate plot that a possible ideal upper limit for learning rate could be 0.1, and we may try decaying learning rate to improve the learning.

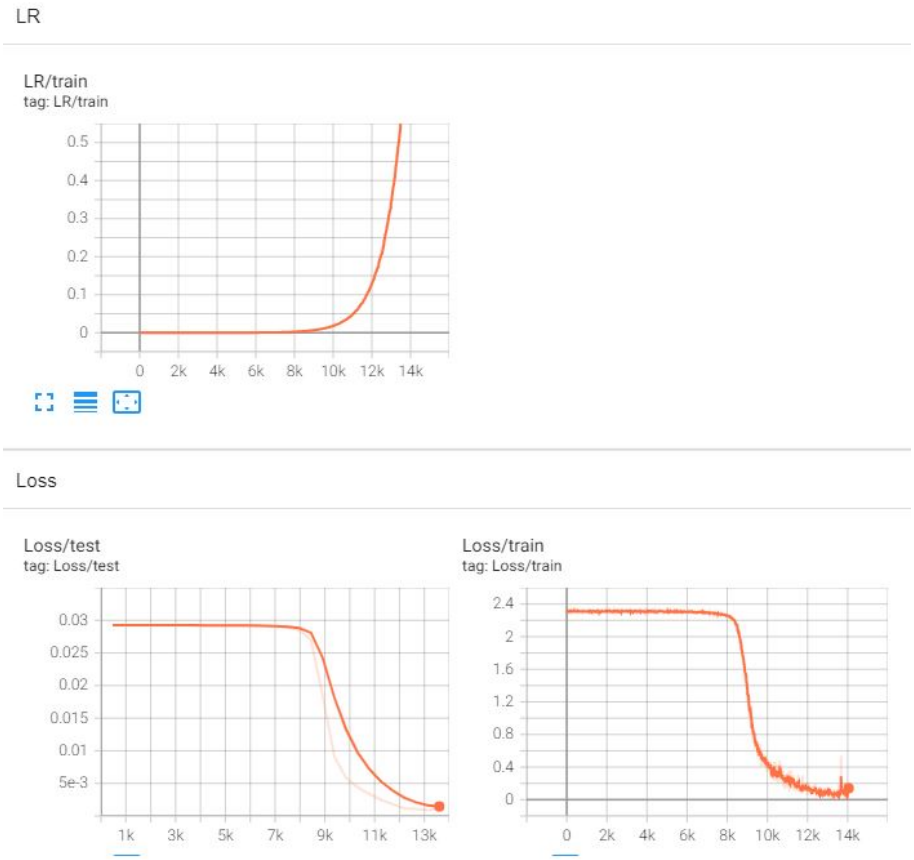

Figure 5: Exponentially Increasing Learning Rate from Low to High

4.2.2 Multistep - milestones=[10,20]

In Figure 6, we can see that while the learning rate is decaying very negligibly, the loss reduces during training. For the test dataset, we can also see the loss reducing. There is some improvements as compared to Figure 5 as we can see that the loss is decreasing at a faster speed during the first 8k data. However, there's still rooms for improvement as we can still some unstability in the loss.

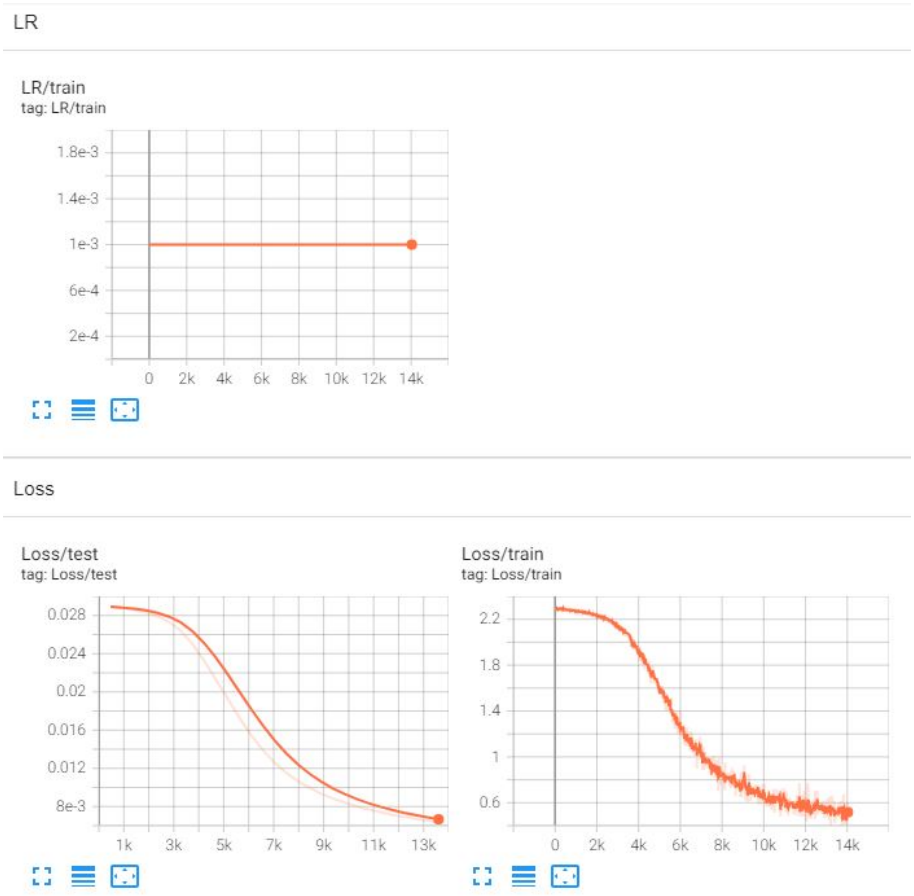

Figure 6: Multistep - milestones=[10,20]

4.2.3 Multistep - milestones=[5,10,15,20,25]

In Figure 7, we attempted adding more milestones to see if we can further decay the learning rate at these milestones. We can see that while the learning rate decays, and in this scenario, it is a sharp decay, the loss during training seem to waver at around the same loss rate. This could mean that the number of milestones set could be unsuitable as the training loss does not seem to improve is wavering at the same point. For the test dataset, we can observe the loss reducing.

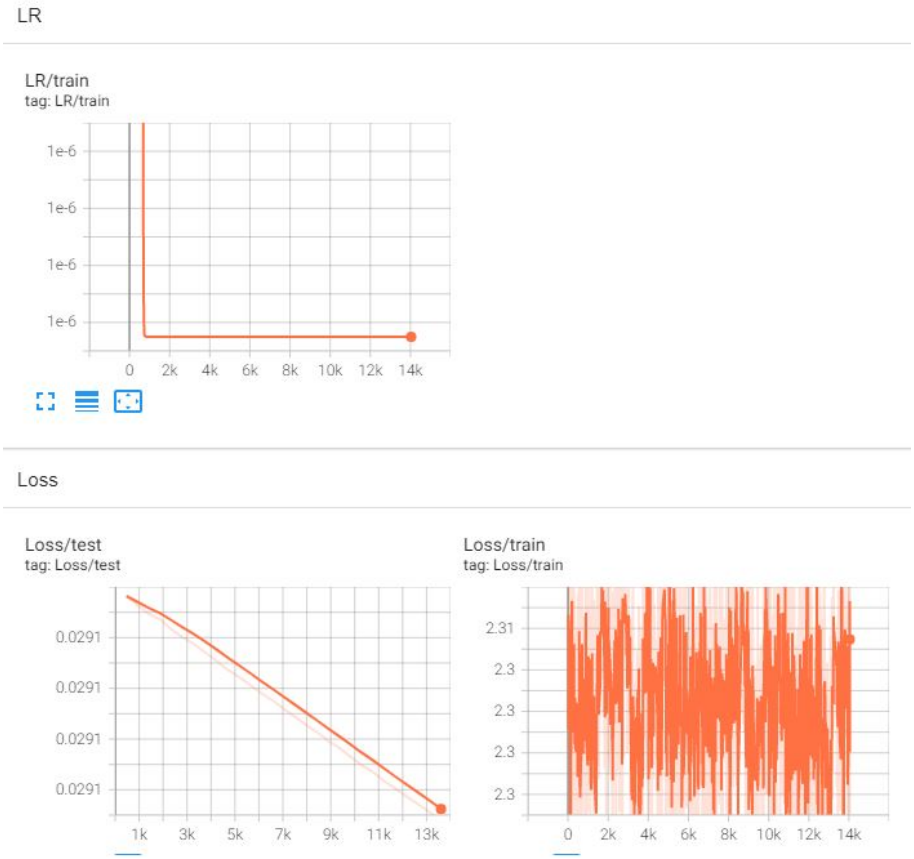

Figure 7: Multistep - milestones=[5,10,15,20,25]

4.2.4 OneCycle - max_lr=3.32

In Figure 8, we can see that the learning rate increases to a peak, before decreasing. The loss during training is observed to follow the same pattern where it first increases exponentially before decreasing. The magnitude of the losses are also very big and is not stable. This suggests that the max learning rate used in the 1cycle learning rate policy might be too large.

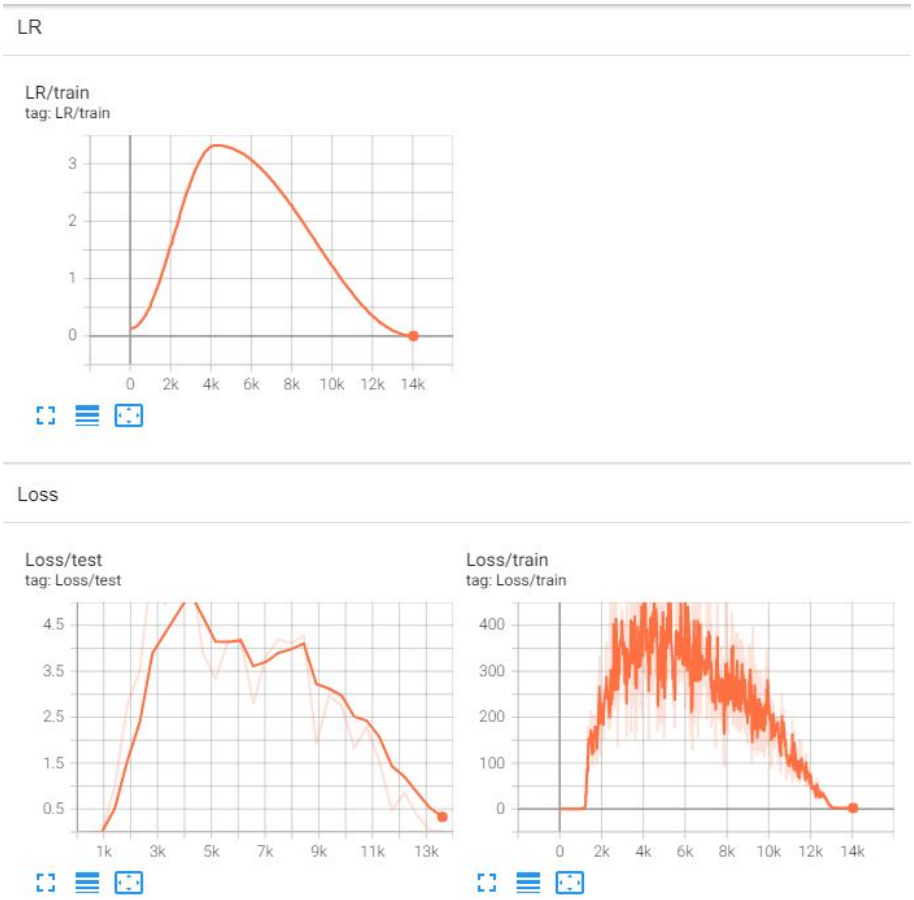

Figure 8: OneCycle - max_lr=3.32

4.2.5 Conclusion

From the experiment conducted in Section 4.2.1, we can observe from the learning rate plot that a possible ideal upper limit for learning rate could be 0.1. We also do know that there needs to be some decay done to the learning rate as there is unstability observed which may observe that the learning rate is too high. Hence, in Section 4.2.2 and 4.2.3, we attempted to decay the learning rate at multiple milestones. There is some improvement for Section 4.2.2 but not for 4.2.3. Next, we try the 1cycle learning rate policy as the learning rate scheduler method which also proves the have a high learning rate would not help reduce loss.

References

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations 2015*, December 2014. doi: https://doi.org/10.48550/arXiv.1412.6980.

L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. August 2017. doi: https://doi.org/10.1080/01621459.2017.1285773.

Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. August 2017. doi: https://doi.org/10.48550/arXiv.1708.03888.

Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *International Conference for Learning Representations 2020*, April 2019. doi: https://doi.org/10.48550/arXiv.1904.00962.