Effects of batch size scaled learning rates and gradual warm up for training deep nets using large mini-batches

Bayrem Kaabachi, Marius Ødum, Marie Knöpfel
Optimization for Machine Learning CS-439, EPFL Lausanne, Switzerland

***Abstract*—Stochastic Gradient Descent (SGD) is an iterative method for large-scale optimization problems in machine learning. In order to train deep neural networks, mini-batches are commonly used to reduce the computation time.**

The aim of this project has been to investigate and assess the challenges in relation to the scalability gains, when using large mini-batches for training deep neural networks. More specifically the project gives a look into the effects of scaling the learning rate to the batch size and adding a gradual warm-up strategy. Throughout the project we show that applying batch size scaled learning rates improves test accuracy significantly for semi-large batch sizes compared to baseline models, while it was challenging to document the advantages of using gradual warm-up due to the utilization of large learning rates.

I. Introduction

Most deep learning applications currently depend on stochastic gradient descent (SGD) and more specifically on its mini-batch variant. It allows parallelization which is a key aspect for fast and efficient training of large scale deep learning models.

We consider a sum structured optimization problem of the form:

$$\mathcal{L}(\mathbf{w}) = \min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{w})$$

where $\mathbf{w} \in \mathbb{R}^d$ denotes the parameters of the model and $f_i : \mathbb{R}^d \to \mathbb{R}$ the cost function of the i-th datapoint, taken from a training set of N elements in total. The mini-batch SGD update is then given as:

$$\mathbf{w}_{(t+1)} := \mathbf{w}_{(t)} - \eta \frac{1}{|B|}\sum_{i\in B} \nabla f_i(\mathbf{w}_{(t)})$$

where $\eta > 0$ denotes the learning rate and B a mini-batch of size $|B|$ sampled from N.

Deep learning consists of large neural networks and large datasets which implies long training times. Mini-batch SGD is a solution to reduce computation time. But to make it more efficient, the mini-batch size should be sufficiently large. This bring some difficulties and adaptation in order to maintain training and generalization accuracy of small mini-batches optimization.

For this project, we put emphasis on the adaptation of the learning rate depending on batch sizes. In recent papers, some of these questions have already been observed [1] [2] [3]. A solution that has been adopted in many studies to improve large batch training performance for a given training computation cost is the linear scaling rule: *"When the mini-batch size is multiplied by k, multiply the learning rate by k."* as stated in Goyal et al., 2017 [1]

If we go back to the theory, the mean value of the mini-batch SGD weight update is given by $-\eta\mathbb{E}\{\nabla f_i(\mathbf{w}_{(t)})\}$. Which means that for a given batch size $|B|$ the expected value of the weight update per unit cost of computation per training example is proportional to $\eta/|B|$. Thus, when increasing the batch size, a linear increase of the learning rate η with the batch size $|B|$ is required to keep the mean SGD weight update per training example constant. [3]

In Goyal et al., 2017 [1], the used very large batch sizes up to 8192. This caused an important loss in generalization performance. They observed that for those numbers, the linear scaling rule was not sufficient and a warm-up strategy was needed. Lower learning rates are used at the start of training to overcome early optimization difficulties because of the network changing rapidly. This allowed them to achieve matching training curves and validation error for both large and small mini-batch SGD without problems in the generalization.

II. Experimental Settings

To observe the changes induced by larger batch optimization, we used the image classification task on the CIFAR-10 dataset. It consists of 60'000 colour images of 32×32 pixels with 50'000 for training and 10'000 for testing. There are 10 different classes, so 6'000 images per class.

The training setup and the models were exclusively build on the PyTorch library. All experiments were also carried out on Google Colab's free GPU service in order to get more computation power[See Appendix].

We did not do any kind of preprocessing on our data, except a normalization of our tensor images. We were mainly interested in comparing the different ways of adapting the learning rates and not finding the best accuracy possible for the different batch sizes.

For our model, we used a simple convolutional neural network that allowed us to output a reasonably good accuracy on which we would be able to compare our different experiments. The implementation uses 3 convolutional blocks, each of them containing two convolutional layers with padding equal to 1 and kernel sizes equal to 3 followed

by a batch normalization for stabilization and a maxpool to reduce the size with a kernel of size 2×2 and stride equal to 2. There is also a 5% dropout between the second and third block to reduce overfitting. Thus, there are in total 6 convolutional layers and the number of channels per layer equal to [3, 32, 64, 128, 128, 256, 256]. Those three blocks are followed by three fully connected layers with two dropouts of 10%. Between each layer, the activation function used is The Rectified Linear Unit (ReLU).

We used the standard SGD optimization without momentum for all of the experiments. It has been shown in previous paper that momentum depends on the batch size. Thus we made the decision as in Masters et al., 2018 [3] to not use any momentum to have a better understanding of our problematic which was the interactions between the learning rate and the batch size only.

To proceed with our analysis, we defined 4 experiments in order to make comparisons between the different batch sizes and methods used.

1) We first computed the standard threshold which is a simple training and test error output of the dataset with the model and optimizer we implemented. The batch-sizes observed are [4, 8, 16, 32] and the learning rate used is $\eta = 0.001$. For each epoch, the implementation was run for 50 epochs.
2) We then proceed to set up the baseline for the large batches to compare the gains of learning rate scaling and the warm-up strategy. We did again a simple training and test error output of the dataset with the model and optimizer implemented. The large batch-sizes observed are [128, 256, 512, 1024, 2048] and the learning rate is again $\eta = 0.001$. The batch-sizes were chosen this way in order to simplify the implementation and analysis of the linear scaling rule of the learning rate with the batch-sizes. Each batch-size was run for a different number of epochs as follow: [100, 150, 200, 300, 400]. We chose to not use a constant number of epochs because we were interested in assessing the time it would get for the batch-size to stabilize on a final accuracy.
3) We implemented the first adaptation for the learning rate which is the linear rule. We simply multiplied our constant learning rate previously chosen by the batch-size, thus for each batch size we would get a different learning rate $lr = |B| \cdot \eta$. We did the usual training and test error output of the dataset with the model and optimizer implemented. We observed the large batches only and each was run for 100 epochs.
4) Finally we added a warm-up to the previous experiment. The warm-up is used for the first 5% of the epochs. We implemented three strategies: an exponential increase and a linear increase of the learning rate until it reaches the lr and a constant low learning rate until it changed at 5% of the epochs number to lr from the scaling rule. We did the usual training and test error output of the dataset with the model and optimizer implemented. We observed the large batches only and each was run for 70 epochs.

III. Results

In fig. 1, the test accuracies over the epochs for the different batch-sizes are shown. As seen the best accuracy is 82% and reached for the batch size equal to 4.

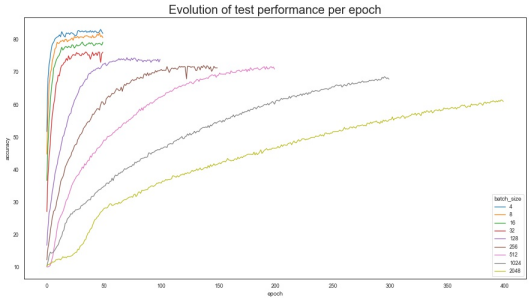

Figure 1. Test accuracy per epoch for each batch-size we implemented

Table I
TIME EVALUATION FOR THE EPOCH AND ACCURACIES FOR EACH BATCH SIZE

Batch size	Epoch to acc.	Time pr. epoch [s]	Time to acc. [s]
4	∼35	79.49	2782.15
8	∼35	46.41	1624.35
16	∼35	27.17	950.95
32	∼35	18.02	630.7
128	∼80	16.96	1356.8
256	∼130	16.23	2109.9
512	∼200	15.88	3176

The results for the three experiments on the large batches are summarised in fig. 2, 3, 4, 5 and 6. Due to the limitation in the use of Google colab, we focused on the exponential strategy for the warm-up and only compared results between the experiment with the learning rate equal to the one for the smalle batches, the scaled learning rate and then the addition of the warm-up.

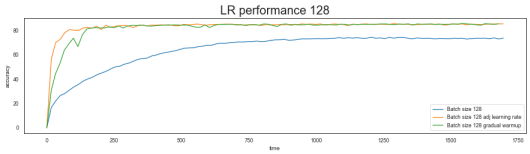

Figure 2. Test accuracy for all experiments for batch-size 128

IV. Discussion

As seen in fig. 1, the final and best accuracies were reached rapidly for the small batches, while the ones for the

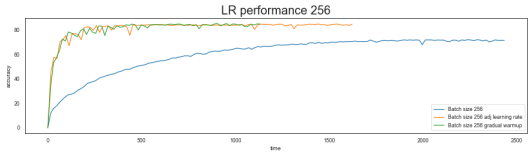

Figure 3. Test accuracy for all experiments for batch-size 256

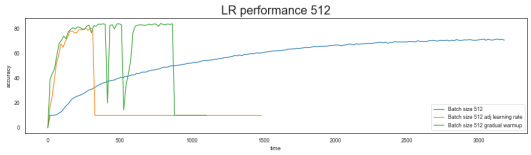

Figure 4. Test accuracy for all experiments for batch-size 512

large batches took much more time lower by a significant amount. In contrast, the larger the batch-size the faster the time to run one epoch. This justified our motivation for this project of trying to find a solution to improve the optimization on the larger batch-size to match the accuracy of the smaller ones.

In fig. 2, 3, 4, 5 and 6, we can clearly see the evolution when growing the batch-size. For each batch-size, we can see some important improvements in the test performance while using the linear scaling for the learning rate. For the warm-up strategy it is not as obvious to see the advantages for the method. It becomes more clear for the very large batches such as 1024 or 2048, where the accuracy are higher and more rapidly obtained.

As seen from fig. 7, the loss is not constantly converging towards a minimum for the larger batch sizes (specifically 512, 1024 and 2048), but instead "jumps" back up at random epochs. This behaviour seems to be attributed to the very large learning rates that arises when training with batch scaled learning rates.

But it is also hard to make assertive conclusions due to some discontinuity in our results, especially for the very large batches. First we noticed some jumps in the accuracies when we used the scaling. We also had diverging losses which implied no test performance starting from a

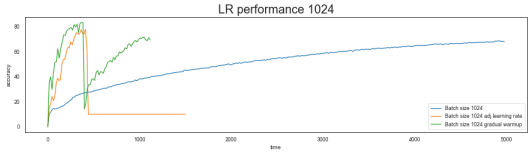

Figure 5. Test accuracy for all experiments for batch-size 1024

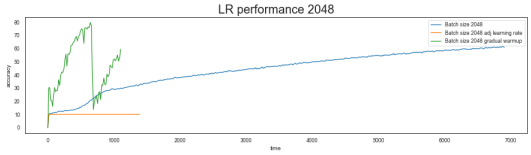

Figure 6. Test accuracy for all experiments for batch-size 2048

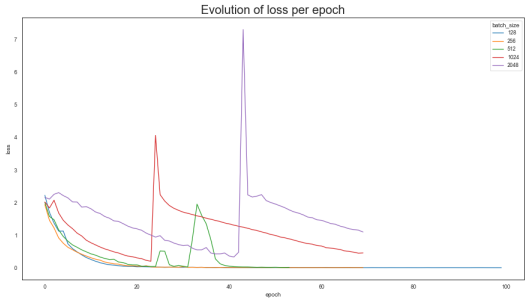

Figure 7. Evolution of the loss per epoch for the large batches

certain number of epoch depending on the batch-size. We can particularly see these two events in fig. 4, 5 and 6. This might be due to the use of a very high learning rate caused by the multiplication in the scaling For the last two batch-sizes we tried, we were optimizing with a learning of approximately 1 and 2 which is very high.

Due to the various abnormal results, especially from the training of the largest batch sizes, it is difficult to draw any conclusion about the effect of the learning rate scaling and gradual warm-up for these batch sizes. It can be argued that for the largest batch sizes another learning rate scaling approach could likely assist in achieving valid results and perhaps even reach accuracies close to the smaller batch sizes with the gradual warm-up. The warm-up actually appears to stabilize the loss during the training of the largest batch sizes, which implies that it is the large learning rate causing the abnormal results.

V. Conclusion

Throughout our research, we found that there is a trade-off between making your mini-batches larger. The gains of increasing the batch size in computation costs gets offset by requiring more gradient updates to converge to full accuracy.

By adjusting the learning rate in large batches and performing gradual warm-up it is possible to reach similar accuracies to the ones we had with mini batches in just as much time or less.

Although we show that it is possible to reach similar performance as mini-batches in some conditions, we also demonstrated that the instability of training with large mini-batches and large learning rates can lead to the model performing poorly or not converging at all.

References

[1] Goyal P., Dollár P., Girshick R., Noordhuis P., Wesolowski L., Kyrola A., Tulloch A., Jia Y., He K., "Accurate, large minibatch sgd: Training imagenet in 1 hour," 2017, https://arxiv.org/abs/1706.02677.

[2] Lin T., Stich S. U., Patel K. K., Jaggi M., "Don't use large mini-batches, use local sgd," 2020, https://arxiv.org/abs/1808.07217.

[3] Masters D., Luschi C., “Revisiting small batch training for deep neural networks,” 2018, https://arxiv.org/pdf/1804.07612.pdf.

APPENDIX

We ran our models on the following hardware:

```
Thu Jun 17 21:10:32 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 8. Appendix: Colab hardware