Experimental Analysis of Training Deep Networks with Very Large SGD Mini-Batches

Mihai David, Tommaso Gargiani, Ana-Arina Raileanu
{mihai.david, tommaso.gargiani, ana-arina.raileanu}@epfl.ch
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—Batch size can have a significant impact when training deep networks with Stochastic Gradient Descent (SGD). We compare the performance of an adapted VGG13 on CIFAR-10 varying the batch size used for SGD. Furthermore, we analyze five methods of ameliorating the effect of very large mini-batches. Through them, for a batch size of 1024, we achieve a reasonable accuracy of 89.62, significantly higher than our baseline accuracy, 39.46.**

I. Introduction

The aim of this report is to gain insights into the impact of the batch size used in Stochastic Gradient Descent (SGD) on network performance. Larger batch sizes are appealing, as they can decrease the time of a training on distributed GPUs, where communication might be a bottleneck [1]. However, it has been theorized that very large batches may degrade the generalization power of a network [2]. Therefore, we evaluate the performance of a network when SGD converges in relation to the batch size, to observe the behavior of the curve on the task of image classification. We want to see whether there is a point where increasing the batch size either brings no improvement or affects the performance of the model. Furthermore, we are interested to see whether this effect holds regardless of the properties of the optimizer, to understand if researchers should consider them when choosing the batch size.

II. Models and Methods

We take a look at a popular network architecture, VGG13 [3], and we assess its performance in relation to the mini-batch size on the task of image classification. The classification task is performed on the well-known dataset, CIFAR-10, consisting of 50000 training images and 10000 test images, part of 10 classes. We further split the training set to obtain a validation set of 10000 images.

The input images, originally in the range $[0, 250]$, are re-scaled to the range $[0, 1]$ and normalized. We augment the training dataset by applying horizontal flips, rotations and translations with a probability of 0.5.

We configure each run through a JSON configuration file, which allows us to easily test various combinations of parameters. We create six experiments, out of which one is considered the baseline. Each experiment consists of six runs (configurations) with different batch sizes: 32, 64, 128, 256, 512, and 1024. We refer to the batch size 32 as our baseline. Also, in order to isolate the interaction between batch size and learning rate, we do not use momentum, as its optimum coefficient is dependent on the batch size [4], [1].

The first experiment, whose results we use as reference for our comparisons, is characterized by a fixed learning rate of 0.01, training the models for a fixed number of epochs, and not using batch normalization. The choice of the learning rate is motivated by the grid-search results on the baseline model, with a batch size of 32. Since we keep, regardless of the current batch size configuration, the learning rate and the number of epochs fixed, we keep the same computational complexity (and a constant number of gradient calculations) [1].

In the second experiment, instead of using a fixed number of epochs for training, we linearly scale the number of epochs based on the batch size. For the batch sizes 32, 64, 128, 256 and 512 we chose 100, 200, 400, 800 and 1600 epochs respectively. We omit testing the largest batch size, 1024, since it would simply require too many epochs. We decide to do so to give more training time for larger batches [5], as they have a higher variance of the weight updates [1]. The goal of this experiment is to see whether allowing more time for larger batch sizes would equalize the results among the different configurations.

The third experiment consists of implementing multiple techniques that have shown to ameliorate the gap between different batch sizes, and which can be successfully applied simultaneously. We use more complex techniques with the hope of achieving similar results without increasing the number of epochs, which increases total time. Starting from the standard VGG13 network, we implement a layer of batch normalization, an idea which has been suggested in previous studies. We add the batch normalization layers only in the features extractor part of the network. Batch normalization has been shown to be beneficial as it reduces the influence the other layers have on each other during parameter update, regardless of the batch size [1]. Furthermore, it seems that on some networks, using batch normalization reduces the difference of the metric between smaller and larger batch sizes. We decided to use the Ghost Batch Normalization implementation [6], which leverages virtual batches to compute the mean and standard deviation on smaller batches, instead of the batch size used for SGD. This implementation reduced generalization error of large batch sizes on various networks [5]. As such, since our baseline model has a batch size of 32, we use Ghost Batch Normalization with a virtual batch size of 32 for all of our experiments. Since we do not vary the number of epochs as

in the previous experiment, we adjust the learning rate using a linear scaling rule, described by Equation 1.

$$\eta = \eta_{\text{baseline}} \cdot \frac{\text{batchsize}_{\text{experiment}}}{\text{batchsize}_{\text{baseline}}} \tag{1}$$

The $\text{batchsize}_{\text{baseline}}$, as stated previously, is 32, with a corresponding learning rate of 0.01. This approach has been used to account for the differences in accuracy and training curves for a larger batch size [4]. Batch normalization allows for larger learning rates to be used [7], supporting our choice to use a learning rate that scales with the batch size.

The fourth experiment builds up on the third one, additionally integrating learning rate scheduler with a warmup strategy [4]. Warmup refers to using a lower learning rate during the first few training epochs. A gradual warmup is slowly increasing the learning rate towards its final value during the first few epochs. After reaching the desired value, a step decay is used for the learning rate. It has been shown that in some cases the gap between small and large batch sizes can be explained by the improper start of the latter, which the warmup aims to fix. We added this experiment with the goal of measuring the effect of the gradual warmup.

The fifth experiment tests a different learning rate scheduler from the third experiment. As shown later in the Results section, although the gradual warmup strategy of the fourth experiment did indeed improve the performance of larger batch sizes, it did not manage to fully recover the performance gap between smaller and larger batch sizes. Therefore, we decided to use the `CyclicLR` learning rate schedule. It implements a policy which cycles the learning rate between two boundaries with a constant frequency, decaying the peak of each cycle throughout time. We set a constant $\text{factor\_lr} = 10$. Then, we compute the lower bound as $\frac{\text{learning\_rate}}{\text{factor\_lr}}$ and the upper bound as $\text{learning\_rate} \cdot \text{factor\_lr}$. Due to unsatisfactory results with a batch size of 1024, we tried setting factor_lr to 5 and 2, with the latter yielding the best results among the three values.

The sixth experiment follows the same setup as the fifth experiment. However, following the observation that a larger batch size benefited from a smaller factor_lr, we come up with a formula to gradually decrease factor_lr (narrowing learning rate bounds) with increasing batch size, as described by Equation 2.

$$\text{factor\_lr} = \text{factor\_lr}_{\text{baseline}} \cdot \sqrt{\frac{\text{batchsize}_{\text{baseline}}}{\text{batchsize}_{\text{experiment}}}} \tag{2}$$

Larger batch sizes have a smaller range of learning rates for which they provide stable results [1], which motivates our choice to scale the bounds.

We compare the loss and accuracy of the models within the same experiment to evaluate the role of the batch size on convergence, and between experiments to assess possible fixes for a large batch size.

III. Results

In each experiment the best model considered is the one with the highest validation accuracy. The results of our first experiment are shown in Figure 1. Overall, the larger the batch size, the lower the validation accuracy. The difference in accuracy after 100 epochs between the smallest (32) and largest (1024) batch size is a whopping 50%. The validation loss follows the same pattern, being lower for the smaller batch sizes. We can also see that, in the later epochs, the model started overfitting for the batch sizes 32 and 64. Fortunately, this overfitting was not much reflected in the validation accuracy, which remained around the same levels.

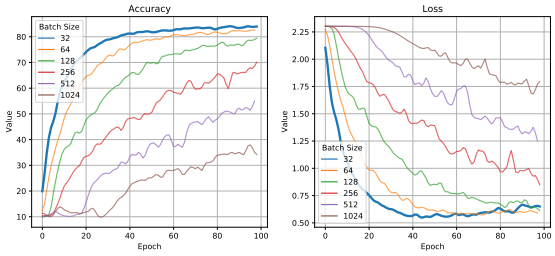

Fig. 1. The baseline validation performance (Experiment 1)

Figure 2 displays the results of the second experiment. We can observe that increasing the number of epochs helped the model generalize substantially better than in the baseline experiment. However, even though the differences between various batch sizes were diminished, this method comes at a significant cost. For the largest batch size we tested, 512, the model has to be trained for 16 times more epochs than our baseline. Put into numbers, training the baseline for 100 epochs took 72 minutes, whereas training the model with a batch size of 512 took close to 20 hours we avoid training on 1024 as it proves to be unfeasible. Increasing the batch size becomes impractical after a certain point. For this reason, we decided to test alternative methods with the goal of maintaining a similar training time.
Eventually, all models started overfitting, as signaled by the rising validation loss. The noise in both accuracy and loss is attributed to the high number of condensed epochs visualized on the plot.

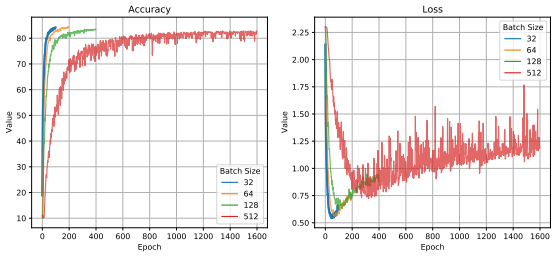

Fig. 2. The validation performance of the network using a number of epochs that scales with batch size (Experiment 2)

As shown in Figure 3, using batch normalization and a linear scaling rule in the third experiment has improved the results

	Batch Size					
	32	64	128	256	512	1024
Baseline	86.37	85.08	82.37	74.40	59.12	39.46
Scaled Epochs	86.58	85.97	85.35	82.79	84.35	-
GhostBatchNorm + Linear Scaling Rule	88.22	88.51	88.44	88.18	81.29	79.07
GhostBatchNorm + Linear Scaling Rule + Gradual Warmup	87.91	88.25	87.79	87.85	87.27	85.97
GhostBatchNorm + CyclicLR	**90.98**	**91.32**	90.78	90.78	90.23	82.79
GhostBatchNorm + CyclicLR + Custom LR Factor	90.88	91.25	**90.96**	**90.82**	**90.60**	**89.62**

TABLE I
TEST ACCURACY FOR ALL EXPERIMENTS AND CONFIGURATIONS

of all the tested batch sizes, virtually erasing any difference between batch sizes 32, 64 and 128 after 100 epochs. This method seems to work best for equalizing the results of reasonably sized batches, but its results degrade after a limit. We continue our search for a method that works well for even larger batch sizes.

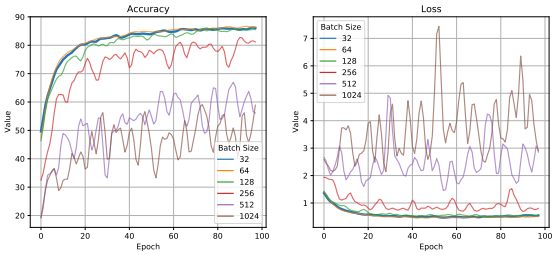

Fig. 3. The validation performance of the network using Ghost Batch Normalization and a learning rated based on a linear scaling rule (Experiment 3)

The gradual warmup of the fourth experiment (see Figure 4) further improved the previous results, managing to substantially increase accuracy even for larger batch sizes, with a constant number of epochs. However, there is still a minor performance gap between the smaller and larger batch sizes. The best results were obtained when using a batch size up to 64. A similar conclusion was provided by researchers in [1]. Due to the step learning rate decay applied at 30, 60, and 90 epochs, with the learning rate being reduced to $\frac{1}{10}$ of its value [4], the models end being trapped in a local minima due to the lower learning rates being used. This motivates our next experiment where, throughout time, we also increase the learning rate, to avoid the aforementioned behaviour.

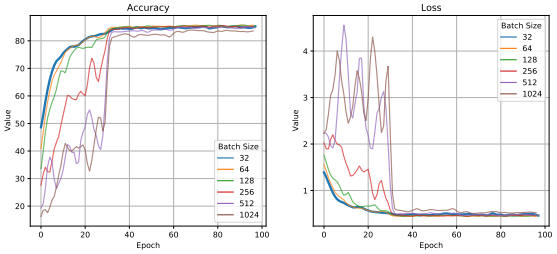

Fig. 4. The validation performance of the network using Ghost Batch Normalization, a learning rated based on a linear scaling rule, and gradual warmup (Experiment 4)

The fifth experiment, in which we changed the learning rate scheduler to `CyclicLR`, improves the results of the models, each one of them achieving better scores than in the fourth experiment for a batch size of up to 512. However, performance is significantly degraded for a batch size of 1024.

In our sixth and final experiment, we aim to fix the difference between the largest batch size, 1024, and the smallest one, using our custom learning rate scaling formula based on the observation that `CyclicLR` benefits from tighter bounds for larger batch sizes. The results can be seen in Figure 5. The differences between all batch sizes are almost unnoticeable. Furthermore, this method allows us to achieve the best results on four of the six batch sizes we tested, as shown in Table III throughout all experiments, without putting a toll on training time.

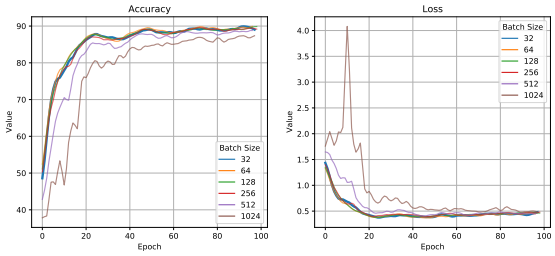

Fig. 5. The validation performance of the network using a Cyclic Learning Rate and a custom formula for the learning rate range (Experiment 6)

IV. CONCLUSION

We have tested five different approaches at reducing the effect of larger batch sizes on accuracy and loss, on the CIFAR-10 dataset, using a VGG13 network. We have shown that the differences between large and small batch sizes can be almost completely erased by scaling parameters with the batch size. Starting with a difference of 46.91 in accuracy between the smallest batch size (32) and the largest one (1024), we reduced it to 1.26, almost 40 times lower. To achieve these results, we combined multiple techniques from recent literature, such as batch normalization and a cyclic learning rate scheduler. On top of that, we developed a custom formula to set the bounds of the above mentioned scheduler based on the batch size, helping us achieve the best results for four out of six batch sizes. As future experiments, we aim at using other datasets, such as CIFAR100 or ImageNet, together with more complex architectures, in order to validate our findings.

REFERENCES

[1] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1804.07612

[2] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1eyO1BFPr

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

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

[5] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1705.08741

[6] N. Bhansali, "Ghost-batchnormalisation-," https://github.com/nixx14/Ghost-BatchNormalisation-, 2020.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: https://arxiv.org/abs/1502.03167