Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes

Takuya Akiba
Preferred Networks, Inc.
akiba@preferred.jp

Shuji Suzuki
Preferred Networks, Inc.
ssuzuki@preferred.jp

Keisuke Fukuda
Preferred Networks, Inc.
kfukuda@preferred.jp

Abstract

We demonstrate that training ResNet-50 on ImageNet for 90 epochs can be achieved in 15 minutes with 1024 Tesla P100 GPUs. This was made possible by using a large minibatch size of 32k. To maintain accuracy with this large minibatch size, we employed several techniques such as RMSprop warm-up, batch normalization without moving averages, and a slow-start learning rate schedule. This paper also describes the details of the hardware and software of the system used to achieve the above performance.

1 Introduction

Training deep neural networks is computationally expensive. Acceleration by distributed computing is required for higher scalability (larger datasets and more complex models) and for higher productivity (shorter training time and quicker trial and error). This paper demonstrates that highly-parallel training is possible with a large minibatch size without losing accuracy on carefully-designed software and hardware systems.

We used the 90-epoch, ResNet-50 [5] training on ImageNet as our benchmark. This task has been extensively used in evaluating performance of distributed deep learning [3, 4, 10]. Table 1 shows the summary of these previous attempts along with our new results. We achieved a total training time of 15 minutes while maintaining a comparable accuracy of 74.9%.

The technical challenge is two-fold; On the algorithm side, we have to design training methods that can prevent loss of accuracy with large minibatch sizes, while on the system side, we have to design stable and practical combinations of available hardware and software components.

Table 1: 90-epoch training time and single-crop validation accuracy of ResNet-50 for ImageNet reported by different teams.

Team	Hardware	Software	Minibatch size	Time	Accuracy
He *et al.* [5]	Tesla P100 × 8	Caffe	256	29 hr	75.3 %
Goyal *et al.* [4]	Tesla P100 × 256	Caffe2	8,192	1 hr	76.3 %
Codreanu *et al.* [3]	KNL 7250 × 720	Intel Caffe	11,520	62 min	75.0 %
You *et al.* [10]	Xeon 8160 × 1600	Intel Caffe	16,000	31 min	75.3 %
This work	Tesla P100 × 1024	Chainer	32,768	15 min	74.9 %

2 Training Procedure for Large Minibatches

We build on the training procedure proposed by [4], and the same settings are used unless otherwise specified. We briefly highlight the differences in this section. For further details, please see Appendix A.

RMSprop Warm-up. We found that the primary challenge is the optimization difficulty at the start of training. To address this issue, we start the training with RMSprop [7], then gradually transition to SGD.

Slow-Start Learning Rate Schedule. To further overcome the initial optimization difficulty, we use a slightly modified learning rate schedule with a longer initial phase and lower initial learning rate.

Batch Normalization without Moving Averages. With the larger minibatch sizes, the batch normalization moving averages of the mean and variance became inaccurate estimates of the actual mean and variance. To cope with this problem, we only considered the last minibatch, instead of the moving average, and used all-reduce communication on these statistics to obtain the average over all workers before validation.

3 Software and Hardware Systems

Software. We used *Chainer* [8] and *ChainerMN* [1]. Chainer is an open-source deep learning framework featuring the define-by-run approach. ChainerMN is an add-on package for Chainer enabling multi-node distributed deep learning with synchronous data-parallelism. We used development branches based on versions 3.0.0rc1 and 1.0.0, respectively. As the underlying communication libraries, we used NCCL version 2.0.5 and Open MPI version 1.10.2. While computation was generally done in single precision, in order to reduce the communication overhead during all-reduce operations, we used half-precision floats for communication. In our preliminary experiments, we observed that the effect from using half-precision in communication on the final model accuracy was relatively small.

Hardware. We used *MN-1*, an in-house cluster owned by Preferred Networks, Inc. designed to facilitate research and development of deep learning. It consists of 128 nodes, where each node has two Intel Xeon E5-2667 processors (3.20 GHz, eight cores), 256 GB memory and eight NVIDIA Tesla P100 GPUs. The nodes are interconnected by Mellanox Infiniband FDR.

4 Experimental Results

For running time and accuracy, the mean and standard deviation from five independent runs are reported. The per-worker minibatch size was 32, and the total minibatch size was 32k with 1024 workers.

Training Time. Using 1024 GPUs, the training time was 897.9 ± 3.3 seconds for 90 epochs, including validation after each epoch. Figure 1 illustrates the average communication time (i.e., all-reduce operations) and time to complete a whole iteration (i.e., forward and backward computation, communication, and optimization) over 100 iterations. Our scaling efficiency when using 1024 GPUs is 70% and 80% in comparison to single-GPU and single-node (i.e., 8 GPUs) baselines, respectively.

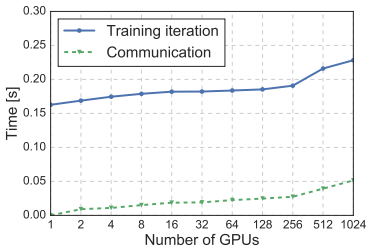

Figure 1: Iteration and communication time for different numbers of GPUs.

Accuracy. After training on 90 epochs using 1024 GPUs with the training procedure designed in Section 2, the top-1 single-crop accuracy on the validation images was $74.94\% \pm 0.09$. As we can observe from Table 1, this accuracy is comparable to that of previous results using ResNet-50. Therefore, it shows that ResNet-50 can be trained on ImageNet with a minibatch size of 32k without severely degrading the accuracy, which validates our claim that training of ResNet-50 can be successfully completed in 15 minutes.

Acknowledgements

The authors thank Y. Doi, G. Watanabe, R. Okuta, T. Kikuchi, and M. Sakata for help on experiments, T. Miyato and S. Tokui for fruitful discussions, and H. Maruyama, R. Calland, and C. Loomis for helping to improve the manuscript.

References

[1] T. Akiba, K. Fukuda, and S. Suzuki. ChainerMN: scalable distributed deep learning framework. *CoRR*, abs/1710.11351, 2017.

[2] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *CoRR*, abs/1511.07289, 2015.

[3] V. Codreanu, D. Podareanu, and V. Saletore. Achieving deep learning training in less than 40 minutes on imagenet-1k. `https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/`, 2017.

[4] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training ImageNet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] T. Tieleman and G. Hinton. Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[8] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *LearningSys*, 2015.

[9] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[10] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer. ImageNet training in minutes. *CoRR*, abs/1709.05011, 2017.

A Details of Training Procedure

A.1 RMSProp Warm-up

Our update rule is a simple combination of momentum SGD and RMSprop [7] (a variant with momentum), defined as follows:

$$
\begin{aligned}
m_t &= \mu_2 m_{t-1} + (1-\mu_2) g_t^2, \\
\Delta_t &= \mu_1 \Delta_{t-1} - \left(\alpha_{\text{SGD}} + \frac{\alpha_{\text{RMSprop}}}{\sqrt{m_t}+\varepsilon}\right) g_t, \text{and} \\
\theta_t &= \theta_{t-1} + \eta \Delta_t.
\end{aligned}
$$

Here, t denotes the current index of iteration. The weights, gradients, momentum, and moving average of the second moment of the gradient at the i-th iteration are represented by θ_i, g_i, Δ_i, and m_i respectively. The inputs are $g_t, \theta_{t-1}, \Delta_{t-1}$, and m_{t-1}, and the outputs are θ_t, Δ_t, and m_t. Hyperparameters are η, μ_1, μ_2, ε, α_{SGD} and α_{RMSprop}: η is the learning rate, μ_1 determines the amount of momentum, μ_2 is the coefficient for the moving average of the gradient second moment, and ε is a small number added for numerical stability. We used $\mu_1 = 0.9, \mu_2 = 0.99$, and $\varepsilon = 10^{-8}$ throughout our experiments. Parameters α_{SGD} and α_{RMSprop} determine the balance between momentum SGD and RMSprop: when $\alpha_{\text{RMSprop}} = 0$, it corresponds to the standard momentum SGD, and when $\alpha_{\text{SGD}} = 0$, it matches RMSprop.

We start with RMSprop (i.e., $\alpha_{\text{SGD}} \approx 0$), and then smoothly switch to SGD (i.e., $\alpha_{\text{SGD}} = 1$). For the transition schedule, we use a function that is similar to the exponential linear unit (ELU) activation function [2] defined as follows:

$$
\alpha_{\text{SGD}} = \begin{cases}
\frac{1}{2}\exp(2(\text{epoch} - \beta_{\text{center}})/\beta_{\text{period}}) & (\text{epoch} < \beta_{\text{center}}) \\
\frac{1}{2} + 2(\text{epoch} - \beta_{\text{center}})/\beta_{\text{period}} & (\text{epoch} < \beta_{\text{center}} + \frac{1}{2}\beta_{\text{period}}) \\
1 & (\text{otherwise})
\end{cases}.
$$

Here, β_{center} and β_{period} are hyperparameters. First, α_{SGD} increases exponentially. At the β_{center}-th epoch, α_{SGD} reaches $\frac{1}{2}$. After that, it increases linearly until the $\beta_{\text{center}} + \frac{1}{2}\beta_{\text{period}}$-th epoch. At the $\beta_{\text{center}} + \frac{1}{2}\beta_{\text{period}}$-th epoch, α_{SGD} becomes 1, and we set $\alpha_{\text{SGD}} = 1$ for the remainder of the training. We set $\beta_{\text{center}} = 10$ and $\beta_{\text{period}} = 5$ throughout our experiments.

We used $\eta_{\text{RMSprop}} = 0.0003$ for the learning rate of RMSprop. Let η_{SGD} be the learning rate of SGD, which will be discussed in the next subsection. To incorporate different learning rates of SGD and RMSprop, we set $\eta = \eta_{\text{SGD}}$ and $\alpha_{\text{RMSprop}} = (1 - \alpha_{\text{SGD}})\eta_{\text{RMSprop}}/\eta_{\text{SGD}}$. One might think that the rule would be simpler if we multiply η_{SGD} to α_{SGD} beforehand, but we should make Δ_t independent from varying learning rates for momentum correction proposed by Goyal *et al.* [4].

A method similar to our RMSprop warm-up is used by Wu *et al.* [9] for a machine translation task. They use the Adam [6] optimizer at the beginning, then switch to SGD. In our preliminary experiments, we found that RMSprop performs better for our task. In addition, Wu *et al.* suddenly switches from Adam to SGD. However, we found that sudden transition severely impacts training and has a negative effect on the final results. Therefore, we designed a smooth transition from RMSprop to SGD. We examined a few transition functions including linear and sigmoid functions. Linear functions have a similar problem at the beginning of the transition. ELU and sigmoid performed similarly, but ELU performs slightly better, so we opted for ELU.

A.2 Slow-Start Learning Rate Schedule

Let η_{base} be the initial learning rate under the linear rule by Goyal *et al.* [4]. Specifically, $\eta_{\text{base}} = 0.1 \cdot \frac{b_{\text{total}}}{256} = 0.1 \cdot \frac{nb_{\text{local}}}{256}$, where n is the number of workers, b_{local} is the local batch size for each worker, and b_{total} is the total batch size among all workers (i.e., $b_{\text{total}} = nb_{\text{local}}$). In our experiments, $n = 1024$ and $b_{\text{local}} = 32$, and thus $\eta_{\text{base}} = 12.8$. Goyal *et al.*'s learning rate schedule is as follows: η_{base} for first 30 epochs, $0.1 \cdot \eta_{\text{base}}$ for the next 30 epochs, $0.01 \cdot \eta_{\text{base}}$ for the following 20 epochs, and $0.001 \cdot \eta_{\text{base}}$ for the last 10 epochs.

To overcome the initial optimization difficulty, we used a slow-start schedule; our learning rate for SGD was $0.5 \cdot \eta_{\text{base}}$ for the first 40 epochs, $0.075 \cdot \eta_{\text{base}}$ for the next 30 epochs, $0.01 \cdot \eta_{\text{base}}$ for the following 15 epochs, and $0.001 \cdot \eta_{\text{base}}$ for the last 5 epochs.