Last Batch Optimization for Distributed DNN Training

Lin Zhang LZHANGBV@CONNECT.UST.HK
Hong Kong University of Science and Technology, Hong Kong

Shaohuai Shi SHAOHUAIS@HIT.ENU.CN
Harbin Institute of Technology, Shenzhen, China

Bo Li BLI@CSE.UST.HK
Hong Kong University of Science and Technology, Hong Kong

Abstract

Data-parallel synchronous optimization has been widely used in distributed DNN training. However, its gradient aggregation communication tasks can easily be pipelined with back-propagation computing tasks, but hard to be pipelined with feed-forward computing tasks, making its scaling efficiency sub-optimal. In this paper, we propose a novel last batch optimization framework to improve the scaling efficiency of distributed training. Specifically, it uses the stale gradient on the last mini-batch to update the model parameter at the current mini-batch, which enables the last batch gradient aggregation (communication) to be fully overlapped with the current batch gradient computation, including both feed-forward and back-propagation passes. We also propose two simple but effective training tricks to reduce the impact of using last batch gradient on convergence. Our experimental results show that last batch optimization can achieve up to $1.83\times$ throughput speedup, while having little accuracy loss compared to synchronous optimization.

1. Introduction

Optimization algorithms, such as stochastic gradient descant (SGD) and Adam [12], lie at the heart of training deep neural networks (DNN) successfully with a wide range of applications [4, 8, 29]. As the training data and model sizes have been rapidly increased, many distributed DNN training techniques have been proposed to accelerate the training process over multiple workers, e.g., data-parallelism [6, 23] and model-parallelism [11, 25]. Among them, *data-parallel synchronous* optimization algorithms such as synchronous SGD are the most popular [6, 30].

Synchronous algorithms partition training data into multiple workers, and all workers calculate the local gradients, and then aggregate the local gradients before they are used to update model parameter in each iteration. Take synchronous SGD (S-SGD) as an example, the update formula is

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{P} \mathbf{g}_t^{(i)}, \tag{1}$$

where $\mathbf{w}_t$ is the model parameter in iteration t, $\mathbf{g}_t^{(i)}$ is the local gradient calculated with local mini-batch on worker i w.r.t. the current $\mathbf{w}_t$, and η is the learning rate. To support efficient gradient aggregation, current training systems [15, 23] typically use *all-reduce* collective operations [2] to aggregate (sum up) all local gradients among P workers. As gradient aggregation tasks can only be pipelined with back-propagation computing tasks [24, 31], the non-overlapped communication

overheads can be significant once given a high communication-to-computation ratio, e.g., training dense models in a limited network bandwidth, resulting in a severe system bottleneck.

Asynchronous algorithms, on the other hand, have been considered to hide more communication overheads, where the local gradient on each worker can update model parameter immediately without waiting for synchronization [9, 21]. For example, the update formula of asynchronous SGD (A-SGD) is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_{t-\tau}^{(i)}, \quad \forall i = 1, \cdots, P, \tag{2}$$

where $\mathbf{g}_{t-\tau}^{(i)}$ is the stale local gradient on worker i w.r.t. a stale copy of model parameter $\mathbf{w}_{t-\tau}$ (which is τ times older than the current parameter). Note the delay τ changes from time to time. A-SGD is typically implemented in the parameter server [14], that is, the worker i pulls the model parameter of $\mathbf{w}_{t-\tau}$ from the server, computes the corresponding local gradient $\mathbf{g}_{t-\tau}^{(i)}$, and then pushes it immediately to the server to update the newest model parameter. However, before this happens, the model parameter has been updated τ times by other workers and becomes $\mathbf{w}_t$. Therefore, the $\mathbf{g}_{t-\tau}^{(i)}$ from worker i has been expired, and using the stale gradient, whose delay τ is *not even fixed*, to model update could cause performance degradation [32]. Meanwhile, as each worker can update model parameter immediately without waiting for gradient synchronization, A-SGD is very hard to implement in the existing well-optimized all-reduce based systems [15, 23].

To overcome these limitations, we propose a novel last batch optimization framework, that uses stale gradient on the last mini-batch (namely last batch gradient) to update model parameter in each iteration. The benefit of last batch optimization for distributed training is that last batch gradient aggregation communication tasks can be *fully* pipelined with the current batch gradient calculation tasks, including all feed-forward and back-propagation passes with or without gradient accumulation [7, 16]. This means more communication overheads can be hidden than synchronous algorithms, and the introduced staleness can be restricted to *fixed 1-step delay* compared to asynchronous algorithms. We conduct extensive experiments to study the efficiency and effectiveness of last batch optimization. The experimental results show that our last batch optimization has achieved $1.09\times$-$1.83\times$ throughput speedups over synchronous optimization in distributed training, and the convergence performance degradation of last batch SGD can be well alleviated by using simple training tricks. It encourages us that last batch optimization can be a promising data parallel paradigm besides synchronous and asynchronous methods.

2. Last Batch Optimization

Last batch algorithms. Last batch SGD (LB-SGD) lies between S-SGD and A-SGD, with the update formula as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^{P} \mathbf{g}_{t-1}^{(i)}, \tag{3}$$

where $\mathbf{g}_{t-1}^{(i)}$ is the last batch gradient calculated with last mini-batch on worker i, w.r.t. the last model parameter $\mathbf{w}_{t-1}$. In LB-SGD, last batch local gradients among P workers can be aggregated efficiently via all-reduce, before they are used to update current model parameter. The staleness of gradient is *fixed* by 1-step delay, which is more reliable than A-SGD [1]. In addition, LB-SGD can be easily extended into other optimizers, e.g., Adagrad [5], NAG [27], and AdamW [18].

Implementation details. Last batch optimization can be implemented very efficiently to pipeline last batch gradient aggregation tasks and current batch gradient calculation tasks, as they have no dependency with each other. We implement our prototype atop PyTorch [20]. The workflow of last batch optimization in each iteration is given as follows: (1) we perform feed-forward (FF) and back-propagation (BP) passes to calculate the current batch gradient; (2) at the same time, we communicate the last batch gradient (which has been stored in a buffer during last iteration) via an all-reduce operation; (3) we use aggregated last batch gradient from the buffer to update model parameter, and then store the calculated current batch gradient into the buffer.

Therefore, the computing tasks in step (1) and the communication tasks in step (2) can be performed simultaneously, so as to *fully* pipeline last batch gradient aggregation and current gradient calculation, including FF and BP passes. As S-SGD can only pipeline gradient aggregation and BP, LB-SGD can hide more communication overheads than S-SGD, having a better system throughput in distributed training. Besides, LB-SGD has two *additional* advantages over S-SGD: (1) it requires only one all-reduce to aggregate all gradient tensors stored together in the buffer, while S-SGD typically calls all-reduce multiple times and introduces extra startup costs [24]; (2) it allows more communication overheads to be overlapped with multiple FF&BP passes using gradient accumulation, while S-SGD can only pipeline communications with the final BP once after the accumulated gradient has been ready (see Figure 1(a)).

Training tricks. Last batch optimization with stale gradient could cause undesirable performance degradation [7, 16]. We propose two simple training tricks to reduce the accuracy loss of LB-SGD. First, we notice that LB-SGD has optimization difficulty in the early training stage (see Figure 1(c)), so we apply *synchronous warmup* that uses S-SGD in the early warmup epochs, and then switch to LB-SGD in the rest epochs. As learning rate in distributed training is often scheduled as a linear warmup followed by the learning rate decay [6], we choose to use S-SGD in the linear warmup stage, and switch to LB-SGD in the learning rate decay stage. Second, we change the original learning rate schedule a bit with *switch decay* to smooth the transition from S-SGD to LB-SGD (see Figure 1(c)). Specifically, we increase the learning rate ratio linearly from 0 to 2 in the warmup stage, and decay the learning rate ratio to 0.4 (by a factor of 5) at the switch moment, and then adjust the learning rate according to a schedule strategy e.g. [17] in the learning rate decay stage.

3. Experiments

We first compare the training efficiency between S-SGD and LB-SGD algorithms on different DNN models to demonstrate that our LB-SGD can achieve higher system throughput than S-SGD in distributed training. Second, we compare the performance between S-SGD and LB-SGD (with training tricks) in training different DNN models and datasets. Finally, we study the effects of last batch optimization with different optimizers, including Adagrad [5], NAG [27], and AdamW [18].

3.1. Training Efficiency

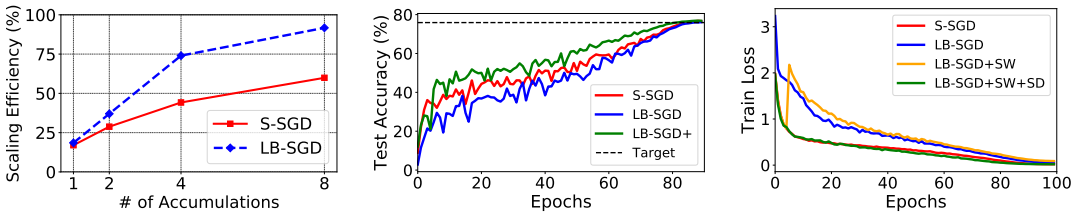
Setup. We compare the training efficiency between LB-SGD and S-SGD on different DNN benchmarks: 3 CNNs on image classification [8, 10, 28], and 2 BERTs on NLP pre-training [4]. We measure the throughput of S-SGD on a popular distributed training system: Horovod-0.21.3 [23] (at PyTorch-1.10). We conduct our experiments in a 64-GPU cluster. It consists of 16 nodes, connected with 10Gb/s Ethernet. Each node has 4 Nvidia GTX 2080Ti GPUs.

Throughput comparison. As seen in Table 1, LB-SGD has better training efficiency than S-SGD. It accelerates distributed training $1.09\times$-$1.83\times$ compared to S-SGD. For three CNN models, S-SGD and LB-SGD achieve 58% and 87% scaling efficiency, respectively. This is because LB-SGD can hide more communication overheads with computing overheads than S-SGD. Indeed, LB-SGD can hide almost all communication overheads in training CNNs, as convolutional layer typically has heavy computation workloads but few model parameters. However, LB-SGD can achieve only 39% and 18% scaling efficiency for BERT-Base and BERT-Large, respectively. We contribute it to the fact that BERTs have dense parameters (e.g., 340M parameters in BERT-Large model), and the 10Gb/s network bandwidth is limited, so that communications cannot be well hidden.

Table 1: System throughput comparison between S-SGD and LB-SGD on 64 GPUs. BS is per-GPU batch size. NP is non-parallel throughput on one GPU. SE_1 and SE_2 are scaling efficiency of S-SGD and LB-SGD, respectively. SP is speedup of LB-SGD over S-SGD.

Model	BS	NP	S-SGD	LB-SGD	SE_1	SE_2	SP
ResNet-50 [8]	64	279.0	13055.1	16688.6	73.1%	93.5%	1.28×
DenseNet-201 [10]	32	150.6	4243.3	7755.4	44.0%	80.5%	1.83×
Inception-v4 [28]	64	181.2	6762.9	10109.2	58.3%	87.2%	1.49×
BERT-Base [4]	64	219.1	4302.8	5492.2	30.7%	39.2%	1.28×
BERT-Large [4]	32	75.8	821.3	894.7	16.9%	18.4%	1.09×

Effects of gradient accumulation. To improve scaling efficiency, one can apply the gradient accumulation technique, i.e., performing FF&BP passes multiple times before the model update, to decrease the communication-to-computation ratio. We compare the scaling efficiency of S-SGD and LB-SGD for training the largest BERT-Large model with different numbers of accumulations. The results are given in Figure 1(a), showing that increasing the number of accumulations can improve the scaling efficiency for both S-SGD and LB-SGD, but the improvement of LB-SGD is much more significant. With 8 accumulations, S-SGD and LB-SGD achieve 60% and 92% scaling efficiency, respectively. The difference lies in the fact that gradient aggregation can be only overlapped with the last BP pass in S-SGD, but it can be fully overlapped with all FF&BP passes in LB-SGD.



(*a*) effects of gradient accum. (*b*) results on ImageNet (*c*) effects of training tricks

Figure 1: Experimental results: (a) effects of gradient accumulation on scaling efficiency, (b) results of training ResNet-50 on ImageNet, (c) effects of training tricks on convergence.

3.2. Convergence Performance

Setup. As we have shown that LB-SGD has better system efficiency than S-SGD, we would like to compare their convergence performance. We select VGG-16 [26] and ResNet-32 [8] on small

Cifar-10 and Cifar-100 datasets [13] with 4 GPUs, and ResNet-50 [8] on large ImageNet [3] dataset with 32 GPUs. We use cosine learning rate schedule [17]. In the first 5 epochs, we use a linear warmup for S-SGD and LB-SGD, and we apply synchronous warmup and switch decay tricks for LB-SGD+. On Cifar-10, we set learning rate to $0.1 \times 4 = 0.4$, per-GPU batch size to 128, epochs to 100, weight decay to 5e-4, and momentum to 0.9. On ImageNet, following [6], we set learning rate to $0.05 \times 32 = 1.6$, epochs to 90, and keep other hyper-parameters the same.

Results on Cifar-10/Cifar-100. The results of the generalization performance of S-SGD and LB-SGD/LB-SGD+ are given in Table 2. On one hand, LB-SGD performs worse than S-SGD with 1.5%-4.3% accuracy loss, and causes even divergence when training VGG-16 on Cifar-10. This validates that using last batch gradient will harm the convergence performance. On the other hand, LB-SGD+ (with training tricks) can reduce the gap to only 0.2%-1.4% accuracy loss. Indeed, we find that S-SGD with the same switch decay trick performs similarly to LB-SGD+, e.g., achieving 93.29% and 92.67% accuracy for training VGG-16 and ResNet-32 respectively on Cifar-10 (which are not reported in Table 2 due to the page size limit). This implies that the minor performance gap between S-SGD and LB-SGD+ could be caused by using switch decay trick. It is of interest to design more effective learning rate schedules for LB-SGD.

Table 2: Test accuracy (%) comparison between S-SGD and LB-SGD/LB-SGD+ algorithms. LB-SGD+ is equipped with synchronous warmup and switch decay training tricks.

Model	Cifar-10			Cifar-100		
	S-SGD	LB-SGD	LB-SGD+	S-SGD	LB-SGD	LB-SGD+
VGG-16 [26]	$93.61_{\pm 0.1}$	$10.05_{\pm 0.0}$	$93.11_{\pm 0.4}$	$73.46_{\pm 0.1}$	$69.14_{\pm 0.4}$	$72.42_{\pm 0.2}$
ResNet-32 [8]	$92.90_{\pm 0.2}$	$91.42_{\pm 0.0}$	$92.71_{\pm 0.1}$	$70.45_{\pm 0.1}$	$68.58_{\pm 0.2}$	$69.00_{\pm 0.3}$

Results on ImageNet. We compare LB-SGD/LB-SGD+ to S-SGD by training a ResNet-50 [8] on the large ImageNet [3] dataset to reach the target 75.9% accuracy [19]. The results given in Figure 1(b) show that LB-SGD performs worse than S-SGD, while LB-SGD+ with two training tricks can surprisingly outperform S-SGD on the large-scale training task. With 90 epochs, S-SGD, LB-SGD and LB-SGD+ achieve 76.7%, 76.1%, and 76.8% test accuracy, respectively.

Effects of training tricks. To study the effects of using synchronous warmup (SW) and switch decay (SD) tricks, we compare the optimization performance of multiple LB-SGD variants by training ResNet-32 on Cifar-10 (results are similar on other models and datasets). As shown in Figure 1(c), LB-SGD optimizes ResNet-32 slower than S-SGD, especially in the early training stage. While LB-SGD+SW can overcome the early stage optimization difficulty, switching from S-SGD to LB-SGD naively will cause a sudden loss spike. We attempt to overcome it with switch decay, i.e., a learning rate decay at the switch moment, so that LB-SGD+SW+SD can optimize closely to S-SGD.

3.3. Training with Different Optimizers

Setup. Finally, we study the performance of different last batch optimizers, including Adagrad [5], NAG [27], and AdamW [18]. We set learning rate to 0.4, 0.04, and 0.004 for NAG, Adagrad, and AdamW, respectively. For AdamW, we set weight decay to 0.05. We run each algorithm for 100 epochs, and use cosine learning rate schedule with a linear warmup in the first 5 epochs, without using synchronous warmup and switch decay tricks for last batch optimization.

Table 3: Test accuracy (%) comparison of synchronous (Sync) and last batch (LB) algorithms.

Dataset	Model	Adagrad [5]		NAG [27]		AdamW [18]	
		Sync	LB	Sync	LB	Sync	LB
Cifar-10	ResNet-32	$89.34_{\pm 0.3}$	$89.45_{\pm 0.1}$	$93.35_{\pm 0.2}$	$92.58_{\pm 0.2}$	$91.20_{\pm 0.0}$	$91.15_{\pm 0.2}$
Cifar-100	VGG-16	$71.42_{\pm 0.4}$	$68.47_{\pm 0.2}$	$74.03_{\pm 0.1}$	$71.90_{\pm 0.3}$	$71.35_{\pm 0.3}$	$68.83_{\pm 0.6}$

The results of different synchronous and last batch optimization algorithms are given in Table 3. It shows that using last batch gradient could have different effects on different optimizers and tasks. For example, last batch adaptive algorithms such as Adagrad and AdamW performs similarly to their synchronous versions when training ResNet-32 on Cifar-10. However, they still have 2.5%-3.0% accuracy loss when training VGG-16 on Cifar-100, which is smaller than 4.3% loss of LB-SGD. Besides, S-NAG and LB-NAG can outperform S-SGD and LB-SGD, respectively. This shows Nesterov's momentum can improve the training performance of synchronous and last batch optimization [7]. It is expected to have different behaviours across optimizers and tasks [22], which reminds us not be limited in SGD and specific tasks to study the effects of last batch optimization.

4. Conclusion

In this paper, we proposed last batch optimization such as LB-SGD for distributed DNN training. We showed that LB-SGD achieved up to $1.83\times$ throughput speedup than S-SGD by fully overlapping last batch gradient aggregation with current batch gradient calculation. Besides, LB-SGD could reach 92% scaling efficiency when training BERT-Large by using gradient accumulation. We studied the effects of LB-SGD on performance convergence, and proposed two training tricks to alleviate the accuracy loss. Finally, we conducted experiments on more optimizers, and their different behaviours remind us to pay more attention to benchmark different last batch optimization algorithms besides LB-SGD. In the future work, we would like to study the effects of last batch optimization with more tasks, develop effective training tricks that work well for different last batch optimizers, and provide a solid convergence analysis.

References

[1] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *ALT*, 2020.

[2] Baidu. *Baidu Ring All-Reduce*, 2017. URL https://github.com/baidu-research/baidu-allreduce.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[5] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *J. Mach. Learn. Res.*, 2010.

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[7] Ido Hakimi, Rotem Zamir Aviv, Kfir Yehuda Levy, and Assaf Schuster. Laga: Lagged allreduce with gradient accumulation for minimal idle time. *2021 IEEE International Conference on Data Mining (ICDM)*, pages 171–180, 2021.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, pages 1223–1231, 2013.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[11] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[14] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX OSDI*, pages 583–598, 2014.

[15] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12): 3005–3018, 2020.

[16] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander G. Schwing. Pipe-sgd: A decentralized pipelined SGD framework for distributed deep net training. In *Annual Conference on Neural Information Processing Systems*, pages 8056–8067, 2018.

[17] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[18] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[19] MLPerf. Mlperf training v1.1 results. https://mlcommons.org/en/news/mlperf-training-v11/, 2021.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[21] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[22] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. In *ICML*, 2021.

[23] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[24] Shaohuai Shi, Xiaowen Chu, and Bo Li. MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 172–180, 2019.

[25] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[27] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[28] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proc. of The 31st AAAI*, 2017.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[30] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. ImageNet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[31] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 181–193, 2017.

[32] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.