Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes

Xianyan Jia[*] **Shutao Song**[*], **Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu**
Tencent Inc.
`xianyanjia,sampsonsong,winsonwhe,slashwang, hudsonrong,hopezhou,felixxie,alexguo, jokeyang,leolwyu,steelchen,gethinhu@tencent.com`

Shaohuai Shi[*], **Xiaowen Chu**
Hong Kong Baptist University
`csshshi,chxw@comp.hkbu.edu.hk`

Abstract

Synchronized stochastic gradient descent (SGD) optimizers with data parallelism are widely used in training large-scale deep neural networks. Although using larger mini-batch sizes can improve the system scalability by reducing the communication-to-computation ratio, it may hurt the generalization ability of the models. To this end, we build a highly scalable deep learning training system for dense GPU clusters with three main contributions: (1) We propose a mixed-precision training method that significantly improves the training throughput of a single GPU without losing accuracy. (2) We propose an optimization approach for extremely large mini-batch size (up to 64k) that can train CNN models on the ImageNet dataset without losing accuracy. (3) We propose highly optimized all-reduce algorithms that achieve up to 3x and 11x speedup on AlexNet and ResNet-50 respectively than NCCL-based training on a cluster with 1024 Tesla P40 GPUs. On training ResNet-50 with 90 epochs, the state-of-the-art GPU-based system with 1024 Tesla P100 GPUs spent 15 minutes and achieved 74.9% top-1 test accuracy, and another KNL-based system with 2048 Intel KNLs spent 20 minutes and achieved 75.4% accuracy. Our training system can achieve 75.8% top-1 test accuracy in only **6.6 minutes** using 2048 Tesla P40 GPUs. When training AlexNet with 95 epochs, our system can achieve 58.7% top-1 test accuracy within **4 minutes** using 1024 Tesla P40 GPUs, which also outperforms all other existing systems.

1 Introduction

With the ever-increasing sizes of datasets and larger deep neural networks (DNNs), training often takes several days if not weeks. For example, training ResNet-50 [12] takes 29 hours using 8 Tesla P100 GPUs. Due to the single machine's limited computing resources, it is natural to distribute the workload to clusters and use supercomputing power to increase the throughput of data flow. A commonly adopted solution is distributed synchronous stochastic gradient descent (SGD) which parallelizes the tasks across machines. Therefore, it is common to use the large mini-batch size to improve the GPU utilization and reduce the communication-to-computation ratio, and thus easy to

[*]Authors contributed equally.

scale out [11, 22]. By using a larger mini-batch size to train a model, fewer updates of the model are needed so that it requires shorter time to train the model with the same number of epochs. However, there are two challenges when using a large mini-batch size across large clusters:

Challenge 1. Larger mini-batch size allows the model to take bigger step size and in turn makes the optimization algorithm progress faster. However, when increasing the mini-batch size to 64K in the ImageNet dataset [8], the test accuracy of ResNet-50 drops [22, 7].

Challenge 2. A distributed training system with data parallelism typically divides batches across GPUs, and requires a gradient aggregation step in each training iteration. This communication step usually becomes the bottleneck of the system when the number of GPUs becomes large. To fully utilize such a distributed training system, we need to improve both the single GPU performance and the system scaling efficiency. To improve the throughput, we need faster computation and more efficient memory bandwidth utilization. To improve the scaling efficiency, we need more efficient collective communication primitives for data exchange.

In this paper, we address the above two challenges. Our contributions are summarized as 1) We successfully scale the mini-batch size to 64K for AlexNet [15] and ResNet-50 training without loss of accuracy. 2) We build a high-throughput distributed deep learning training system which contains two main features to improve the single GPU performance with half-precision and the system scaling efficiency with the optimized all-reduce collective.

2 Related Work

Goyal et al. [11] train the ResNet-50 model with a large mini-batch size of 8K over 256 Tesla GPUs and finish the training process within one hour. You et al. [22] further increase the mini-batch size of ResNet-50 from 8K to 32K by using the Layer-wise Adaptive Rate Scaling (LARS) algorithm. Akiba et al. [2] demonstrate the training of ResNet-50 in 15 minutes with a mini-batch size of 32K over 1024 Tesla P100 GPUs. Similar work have also been demonstrated in [7][19]. However, all the above research towards large-batch training either fail to scale to more nodes and more GPUs with larger mini-batch size, or trade accuracy loss for better performance. Training with low-precision introduces a tradeoff of the number-of-bits used versus the statistical accuracy. Micikevicius et al. [17] propose several techniques including loss-scaling and mixed-precision training for preventing the loss of critical information. However, they have not applied these techniques with large-batch training strategies such as LARS to achieve better performance. To reduce the communication overhead when the number of nodes increases in distributed training, Baidu [9] introduces the ring-based all-reduce algorithm [4] to deep learning. [3][23] further improves the system throughput. However, the original all-reduce version is low in bandwidth utilization when the data size is small or the number of nodes is large. The IBM's PowerAI Distributed Deep Learning (DDL) system [6] has deployed a multi-dimensional ring algorithm to improve the scaling efficiency, and it scales to 512 GPUs with scaling efficiency of 95% on ResNet-50. However, it could not be optimal when scaling to more GPUs due to the impact of the startup time of communications.

3 System Overview

At a high level, our system contains the following three modules: 1) input pipeline module; 2) training module; and 3) communication module.

Input pipeline module: In the procedure of SGD, each worker needs to fetch the data from disk to CPU memory, and then transfers the data to GPU memory. In order to reduce both the CPU and the GPU idle time, one should pipeline the data reading with the computation on the GPU. The module first allocates a data buffer queue for GPU computation, and then invokes multiple CPU threads to read the training data from the disk to the buffer. In each iteration, each GPU worker only needs to read the data from the buffer, which overlaps the I/O operation with the GPU computation.

Training module: The training module contains the weights and gradients management to support mix-precision training. Given a DNN model, the module constructs the weights and gradients of the model. We maintain two copies of the DNN model weights, one with single-precision (FP32) and another with half-precision (FP16). The FP32 one is updated iteratively based on the calculated gradients, while the FP16 one is used to do the calculation of feed-forward and backward propagation.

For the gradients, they are allocated with the FP16 format, and calculated iteratively with SGD. Before being updated to the weights, the FP16 gradients should be aggregated across multiple workers. Compared to the FP32 gradients, the size of data communication is reduced by half.

Communication module: The communication module maintains a message queue that stores the information about which layers should be aggregated across multiple workers. When the queue is not empty, it fetches the message from the head to invoke the all-reduce operation on the specific tensors (the gradients of DNN layers) and decides which all-reduce algorithm should be used according to the tensor size and system configurations (e.g., the number of workers and the bandwidth of interconnection). The communication module mainly makes two decisions to achieve better performance: 1) Tensor fusion: When there are many small tensors to be communicated, it is better to fuse these small tensors to one large tensor which only needs to be communicated once. 2) All-reduce algorithm selection: With different tensor sizes and system configurations, the original ring-based all-reduce algorithm could be sub-optimal. We propose a hierarchical all-reduce algorithm. At runtime, our system uses a hybrid all-reduce algorithm to decide which all-reduce algorithms (the hierarchical all-reduce and the original ring-based all-reduce) to use according to different tensor sizes.

4 System Implementation and Optimization

4.1 Mixed-precision training with LARS

As Micikevicius et al. [17] have mentioned, the motivation of using half-precision (FP16) in the training phase is to release the memory bandwidth pressure as well as increase arithmetic throughput. Orthogonal to half-precision training, You et al. [21] first propose LARS to enable larger mini-batch size for distributed training. The algorithm introduces a local learning rate for each layer (as shown in Equation 1), which is the ratio of the L2-norm of weights and gradients weighted by a LARS coefficient η.

$$\Delta w_t^l = \gamma \cdot \eta \cdot \frac{w^l}{\bigtriangledown L(w^l))} \cdot \bigtriangledown L(w_t^l). \tag{1}$$

Gradients are multiplied with its adaptive local learning rate. A natural choice is to combine half-precision training with LARS to achieve larger mini-batch size with scalability. However, a naïve implementation would introduce several problems because using LARS directly on half-precision training will cause the computed learning rate to be out of the dynamic range of the IEEE half-precision format, and thus cause the gradients to vanish and stall the training process.

The mixed-precision technique has been used to solve the zero gradients problem, but it would also loss accuracy when scaling to a large mini-batch size. As LARS makes training with large mini-batch sizes possible, we propose a training strategy which uses mixed-precision training with LARS. In the proposed strategy, the operations in forward and backward propagation are performed in FP16, while the weights and gradients are cast to single-precision (FP32) format before applying LARS in weight update and cast back to FP16 afterwards.

4.2 Improvements on model architecture

We improve the model architecture to keep the model accuracy from the following two aspects when training with a large mini-batch size: 1) eliminating weight decay on the bias and batch normalization; and 2) adding a proper batch normalization (BN) layer for AlexNet.

Weight decay [16] is a commonly-used strategy to achieve better model generalization by adding a regularization term to the loss function as follows:

$$E(w) = E_0(w) + \frac{1}{2}\lambda \sum_i w_i^2, \tag{2}$$

where $E_0(w)$ is the original loss function, and $\frac{1}{2}\lambda \sum_i w_i^2$ is the L2 regularization term. If gradient descent is used for learning, the last term of the loss function leads to a new term $-\lambda w_i$ in the gradients update:

$$w_i^{t+1} = w_i^t - \eta \frac{\partial E}{\partial w_i^t} - \lambda w_i^t. \tag{3}$$

The above equation applies to both bias and weight parameters. However, regularizing the bias parameters could introduce underfitting, so it is common to only penalizes the weights of the affine transformation at each layer and leaves the biases unregularized [10]. Therefore, in the training with a large mini-batch size, we also leave the bias parameters unregularized. Moreover, the weight decay of L2 regularization is also applied to the trainable parameters of BN layers. The two parameter sets of BN are β and γ, and the update formula is as follows:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2 + \epsilon}}, \text{ and } y_i \leftarrow \gamma x_i + \beta \equiv BN_{\gamma,\beta}(x_i), \tag{4}$$

where β and γ are two trainable parameters, $\mu_{\mathcal{B}}$ is the mean of mini-batch and σ^2 is the variance of mini-batch. What we have observed in our training for AlexNet is that if we also leave two parameter sets β and γ in BN unregularized, the model achieves better convergence.

As Goodfellow et al. [10] have noted, the reason that the model without regularizing the parameters of BN layers achieves better convergence could be that b, β and γ usually have much less parameters compared to the weights W (for AlexNet model, b, β and γ parameters only amount to 0.02% of all parameters), which means that leaving them unregularized would not give too much variance and regularizing could instead introduce a significant amount of underfitting. For AlexNet, we get around 1.3% improvement in accuracy from 55.8% to 57.1% with the same number of epochs. The slight improvement on convergence comes from the reduced computations in L2 regularization.

Batch normalization is generally better than local response normalization (LRN) in generalizing deep models [13][21]. Replacing LRN layers in AlexNet with BN (AlexNet-BN) can improve the accuracy [13]. However, when scaling to a mini-batch of 64K, the AlexNet-BN model can only achieve the top-1 accuracy of 57.1%, while the baseline is 58.8%. By analyzing the parameters and feature map distributions, we find that the feature map distribution after Pool5 has a large variance and maximum values during the training. The significant change of feature scaling makes the training difficult. This motivates us to insert another BN layer after Pool5 to rescale the feature map. The variance and maximum values are significantly reduced after inserting a BN layer. The refined-AlexNet-BN model reaches 58.8% top-1 accuracy with a mini-batch size of 64K in the 95-epoch training.

4.3 Improvement on communication strategies

In a cluster with p GPUs, the ring-based all-reduce algorithm splits the data on each GPU into p chunks and do the reduction in $p-1$ iterations [20]. Its time cost can be modeled as Equation 5 whose first term is the startup time of communication. The startup time is determined by the number of GPUs and is not related to the message size M. To reduce the impact of the startup time, we propose two strategies: tensor fusion and hierarchical all-reduce. And wee finally use hybrid all-reduce which combines original all-reduce and hierarchical all-reduce to further improve the efficiency.

$$t_c = 2(p-1)a + 2\frac{p-1}{p}bM + \frac{p-1}{p}cM. \tag{5}$$

Tensor fusion. Usually, gradient tensor sizes are variant for different layers in a DNN, and convolutional layers are relatively small. Sending too many small tensors in the network could not only under-utilize the bandwidth but also increase the latency. According to Equation 5, aggregating one message with a size of $M_1 + M_2$ is faster than sending the two messages (M_1 and M_2) separately. Therefore, the core idea of tensor fusion is to pack multiple small size tensors together before all-reduce to eliminate some startup times. One can use the optimal tensor fusion strategy to achieve the largest overlap between gradient communication and backward propagation [18].

Hierarchical all-reduce. From Equation 5, the time cost also has a relation with the number of GPUs p. When p becomes too large (e.g., 1024), the startup time has also large contribution to the time cost. Therefore, we propose a hierarchical all-reduce algorithm to reduce p involving the ring-based all-reduce operation. The hierarchical all-reduce algorithm splits all the GPUs into groups and invokes three-step operations with intra-group and inter-group collectives to finish the task of the original all-reduce algorithm. p GPUs are grouped into p/k groups and each group contains k GPUs, the three-step operations are: 1) Reduction: Each group invokes a reduction in parallel. 2) All-reduction: The master GPU of each group constitutes a communication ring (with p/k GPUs), and a ring-based all-reduce collective is invoked on the ring. 3) Broadcast: The master GPU in each group broadcasts the messages to all the other GPUs in the same group. Note that the startup time cost of the all-reduce collective is reduced from $2(p-1)a$ to $2(p/k-1)a$.

Table 1: Compare AlexNet training with different teams

Team	Batch	Hardware	Software	Top1 Acc	Time
You et al. [22]	512	DGX-1 station	NVCaffe	58.8%	6h 10m
You et al. [22]	32K	CPU $\times$ 1024	Intel Caffe	58.6%	11min
This work	**64K**	Tesla P40 $\times$ 512	TensorFlow	**58.8%**	**5m**
This work	**64K**	Tesla P40 $\times$ 1024	TensorFlow	**58.7%**	**4m**

Table 2: Compare ResNet-50 training with different teams

Team	Batch	Hardware	Software	Top1 Acc	Time
He et al. [12]	256	Tesla P100 $\times$ 8	Caffe	75.3%	29h
Goyal et al. [11]	8K	Tesla P100 $\times$ 256	Caffe2	76.3%	1h
Cho et al. [6]	8K	Tesla P100 $\times$ 256	Torch	75.0%	50min
Codreanu et al. [7]	32K	KNL $\times$ 1024	Intel Caffe	75.3%	42min
You et al. [22]	32K	KNL $\times$ 2048	Intel Caffe	75.4%	20min
Akiba et al. [2]	32K	Tesla P100 $\times$ 1024	Chainer	74.9%	15min
This work	**64K**	Tesla P40 $\times$ 1024	TensorFlow	**76.2%**	**8.7m**
This work	**64K**	Tesla P40 $\times$ 2048	TensorFlow	**75.8%**	**6.6m**

Hybrid all-reduce. Hierarchical all-reduce can bring performance gain in some cases, however, it also performs worse than the original all-reduce algorithm when the size of message is large enough. To enjoy the best of both worlds, we use a hybrid strategy in our system. We set a parameter η to represent the size of the tensor to aggregate in bytes to determine which algorithm (hierarchical or original all-reduce) to use.

5 Experimental Results and Analysis

5.1 Experimental settings

Models. We choose AlexNet and ResNet-50 for our experiments because they represent two typical types of CNNs. The parameter size of AlexNet (64M) is around 2.5 times as ResNet-50 (25M), while the computation of ResNet-50 (4 GFLOPs) is around 5.6 times as AlexNet (727 MFLOPs). The baseline top-1 test accuracies of AlexNet and ResNet-50 are 58.8% [22] and 75.3% [12] respectively.

Dataset. The chosen two CNNs are evaluated in the ImageNet dataset. Both models are trained with about 1.28 million training images and evaluated with 50,000 validation images by the top-1 test accuracy on the 1000-class classification task. Images are stored in the format of TFRecord[2]. In all our experiments, we use data augmentation offered in TensorFlow benchmark [3].

Software. Our training system is built on TensorFlow [1]. The communication libraries are NCCL2 and OpenMPI. The CUDA version is 9.2 and the operating system is CentOS-7.2.

Hardware. The experimental GPU cluster includes 256 nodes, and each node contains 8 Nvidia Tesla P40 GPUs that are interconnected with PCIe-3.0. Each server has an Mellanox ConnectX-4 100Gbit Ethernet network card which supports RDMA.

5.2 Overall experimental results

For AlexNet, You et al. [22] can finish the ImageNet training with a mini-batch size of 32K in 11 minutes using 1024 Intel CPUs, while we can train the AlexNet model 2.75 times faster by using 1024 Nvidia Tesla P40 GPUs. To our best knowledge, we create the fastest training time of 95-epoch AlexNet on the ImageNet dataset in 4 minutes with 1024 Tesla P40 GPUs.

[2]The converting method is provided here: `https://github.com/tensorflow/models/blob/master/research/inception/inception/data/buildimagenetdata.py`

[3]`https://github.com/tensorflow/benchmarks/blob/master/scripts/tf_cnn_benchmarks/preprocessing.py`

For ResNet-50, our system finishes the 90-epoch training in 8.7 minutes with 76.2% top-1 test accuracy over 1024 Tesla P40 GPUs and 6.6 minutes with 75.8% top-1 test accuracy over 2048 Tesla P40 GPUs. Compared to Akiba et al. [2], our work saves around 40% cost with similar hardware but much shorter training time. Compared to He et al. [12]'s work which uses 8 GPUs, we achieve more than 248x speedup. Using the same number of 1024 GPUs, we achieve 1.61 times faster and higher top-1 test accuracy than the work in [2]. Note that for ResNet-50 training, we adopt half-precision communication during the all-reduce gradients aggregation phase due to its reduced bandwidth usage.

5.3 Training speed and scalability

Throughput of mixed-precision. To improve the throughput of GPU computation, we use FP16 in the forward and backward propagation, which are two most time-consuming steps during training. Using mixed-precision training can speedup single-node performance of ResNet-50 from 172 images/second to 218 images/second.[4]

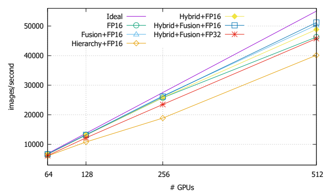

Figure 1: Throughput comparison on AlexNet with a mini-batch size of 128 per GPU

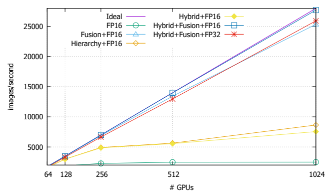

Figure 2: Throughput comparison on ResNet-50 with a mini-batch size of 64 per GPU

Scalability. The scaling performance comparisons of AlexNet and ResNet-50 are shown in Fig. 1 and Fig. 2 respectively. For AlexNet, it can be seen that FP16 performs better than FP32 since FP16 communicates a half size of message compared to FP32. Tensor fusion would improve the scaling efficiency from 70% to 81% without other optimization using FP16 on 512 GPUs. Without tensor fusion, only using hierarchical all-reduce results in a worse performance, but the proposed hybrid all-reduce achieves about 89% scaling efficiency. Combined with FP16 and tensor fusion, the hybrid all-reduce would achieve 91.4% scaling efficiency with 512 GPUs on the AlexNet model. Regarding ResNet-50, the mini-batch size for each GPU is 64 running on 1024 GPUs (results in a valid batch size of 64K), the throughput shown in Fig. 2 shows that our customized all-reduce has the highest scaling efficiency. The scaling efficiency of 1024 GPUs (8 GPUs per node, and totally 128 nodes) compared to single-node (8 GPUs) is up to 99.2%, which is very close to the optimal scaling efficiency. When comparing the scaling efficiency before and after optimization, we can see the improvement is significant just like in the above analysis of AlexNet. By using all optimization techniques (FP16 format, tensor fusion and hybrid all-reduce) in communication, we achieve the scaling efficiency of 99.2% on the 1024-GPU cluster.

In order to compare with the previous work in [2], we run the benchmark using the same configuration as theirs (set the mini-batch size as 32 per GPU on the 1024-GPU cluster). The experimental result shows that our system achieves a scaling efficiency of 87.9%, which is 7.9% higher than their 80%. When per GPU mini-batch size is 32, it is harder to scale out. Because the smaller mini-batch size often leads to faster computation, and thus higher communication-to-computation ratio, which causes the communication easily become the bottleneck of the system. Due to our efficient communication strategies, we have achieved higher scaling efficiency than the state-of-the-art with the same mini-batch size. Due to limited space, the full version of our experiment analysis can be found in [14].

Although we discuss several of our optimizations only in the scope of training convolutional networks on ImageNet, the mixed-precision strategy, tensor fusion and hybrid all-reduce are model agnostic and could be applied to other types of neural networks to increase the efficiency of the training system in the distributed environment.

[4]Note that the tested P40 GPU does not have native support for half-precision computation, hence the speedup of half-precision is mainly contributed by the software optimization in cuDNN [5]. One could achieve much higher throughput by using GPUs that support half-precision computation in the hardware level (e.g., Tesla P100 or V100 GPUs).

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[3] Ammar Ahmad Awan, Khaled Hamidouche, Jahanzeb Maqbool Hashmi, and Dhabaleswar K Panda. S-Caffe: co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters. *Acm Sigplan Notices*, 52(8):193–205, 2017.

[4] M. Barnett, L. Shuler, R. van de Geijn, S. Gupta, D. G. Payne, and J. Watts. Interprocessor collective communication library (InterCom). In *Proceedings of IEEE Scalable High Performance Computing Conference*, pages 357–364, May 1994. doi: 10.1109/SHPCC.1994.296665.

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[6] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. Powerai ddl. *arXiv preprint arXiv:1708.02188*, 2017.

[7] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[9] Andrew Gibiansky. Bringing hpc techniques to deep learning. 2017. URL `http://research.baidu.com/bringing-hpc-techniques-deep-learning`.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[11] 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.

[12] 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.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[14] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992. URL `http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf`.

[17] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

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

[19] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[20] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[21] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32K for ImageNet training. *CoRR*, abs/1708.03888, 2017. URL `http://arxiv.org/abs/1708.03888`.

[22] Yang You, Zhao Zhang, C Hsieh, James Demmel, and Kurt Keutzer. ImageNet training in minutes. *CoRR, abs/1709.05011*, 2017.

[23] 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 *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, pages 181–193. USENIX Association, 2017.