Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2020 SCPE. Volume 21, Issue 2, pp. 323–336, DOI 10.12694:/scpe.v21i2.1727

A DISTRIBUTED NEURAL NETWORK TRAINING METHOD BASED ON HYBRID GRADIENT COMPUTING

ZHEN LU[*], MENG LU[†], AND YAN LIANG[‡]

Abstract. The application of deep learning in industry often needs to train large-scale neural networks and use large-scale data sets. However, larger networks and larger data sets lead to longer training time, which hinders the research of algorithms and the progress of actual engineering development. Data-parallel distributed training is a commonly used solution, but it is still in the stage of technical exploration. In this paper, we study how to improve the training accuracy and speed of distributed training, and propose a distributed training strategy based on hybrid gradient computing. Specifically, in the gradient descent stage, we propose a hybrid method, which combines a new warmup scheme with the linear-scaling stochastic gradient descent (SGD) algorithm to effectively improve the training accuracy and convergence rate. At the same time, we adopt the mixed precision gradient computing. In the single-GPU gradient computing and inter-GPU gradient synchronization, we use the mixed numerical precision of single precision(FP32) and half precision(FP16), which not only improves the training speed of single-GPU, but also improves the speed of inter-GPU communication. Through the integration of various training strategies and system engineering implementation, we finished ResNet-50 training in 20 minutes on a cluster of 24 V100 GPUs, with 75.6% Top-1 accuracy, and 97.5% GPU scaling efficiency. In addition, this paper proposes a new criterion for the evaluation of the distributed training efficiency, that is, the actual average single-GPU training time, which can evaluate the improvement of training methods in a more reasonable manner than just the improved performance due to the increased number of GPUs. In terms of this criterion, our method outperforms those existing methods.

Key words: Deep learning, gradient descent, mixed precision computing, distributed training

AMS subject classifications. 97R40, 97P50, 97P10

1. Introduction.

In recent years, deep learning technology has a far-reaching impact on the fields such as computer vision [1, 2, 3, 4, 5], speech recognition [6, 7], and natural language processing [8, 9]. The rapid development of deep learning is not only due to the innovation of algorithms, but also benefited from the substantial improvement of computing power, which has become the core competitiveness of deep learning technology. In the field of computer vision, for example, several major technological breakthroughs are driven by larger neural networks and larger datasets. Neural networks have increased from 8 layers [1] and 22 layers [4] to hundreds of layers [5]. There are academic datasets of more than one million samples [10], and the industry has used datasets of billion samples [11]. Only with the support of computing power, researchers can complete such scale training tasks in a reasonable time. At present, the commonly used computing power promotion method in the industry is based on distributed data-parallel training [12, 13, 14, 15, 16]. Due to the limited computing resources of a single GPU, data is allocated to multiple GPUs, and the data of each GPU is combined to form a larger batch, that is, it has a larger batch size, which enables training across GPUs in an inter-computer manner. By training at the same time, the throughput of computation can be increased and the training speed can be improved. However, distributed data-parallel training is still an open research field, which mainly faces three problems.

Problem 1: Due to the optimization [12] and generalization [17, 18] difficulty of large batch training, the training accuracy will be lower than that of small batch. Gradient estimation on larger batch is more accurate, which allows a larger step size in the gradient descent, making the process of the optimization algorithm faster. However, as described in [19], when the batch size increases to 64K, ResNet-50 validation accuracy drops from 75.4% to 73.2%.

[*]Guangzhou College of South China University of Technology, Guangzhou 510800, China (luzhenaaa@126.com).
[†]China mobile (Suzhou) software technology co., Suzhou 21500, China (lumeng@cmss.chinamobile.com).
[‡]Corresponding author, Alibaba Group, Guangzhou 510335, China(liangyan_709@163.com).

Problem 2: Distributed data-parallel training divides the gradient computing into multiple GPUs, and gradient need to be synchronized and aggregated. With the increase of the number of GPUs, the number of communications between multiple GPUs and the total amount of communication data increase, making it difficult to maintain the linear throughput scaling. Especially for the model with a large amount of parameters, there are too many gradient data to be synchronized, making the communication a bottleneck and affecting the training speed. If the GPUs are distributed on multiple machines, inter-machine communication will be more difficult than the intra-machine communication, which will result in lower scaling efficiency.

Problem 3: We define the convergence rate of training as the reciprocal of the number of epochs completed when converging. That is to say, the less epochs are used when convergence, the faster the convergence rate will be. Because the current evaluation criterion of distributed training speed is the total time consumed for fixed epochs (the default is 90 epochs), there are few researches and reports on convergence rate in the industry.

In this paper, an efficient distributed neural network training method is proposed for addressing the above three problems. Our contribution includes:

1. A hybrid gradient descent method is proposed, which is a new warmup scheme on the AdaBound algorithm [20] combined with the linear scaling SGD algorithm [12]. This method can effectively improve the convergence rate without losing the training accuracy, and fill the blank in the research area of improving the convergence rate in the distributed system.
2. In this paper, the hybrid gradient precision computing is used, and the FP16 numerical precision is used in single-GPU gradient computing and multi-GPU gradient synchronization. At the same time, many strategies to avoid numerical overflow are proposed. Without sacrificing the training accuracy, the training speed of single-GPU is improved, and the speed of multi-GPU synchronous aggregation is also improved because the communication volume is compressed.
3. We have constructed a real distributed training system by carefully integrating various training strategies, such as batch normalization (BN) [21], layer-wise adaptive rate scaling (LARS) [16] and so on. In our training system, we have accomplished the ResNet-50 training in 20 minutes, with the Top-1 accuracy being 75.6%, and the scaling efficiency more than 97%.

In order to compare with the existing distributed training methods in the industry, we propose a new performance evaluation criterion. At present, the commonly used evaluation criterion is the training time of training the classic ResNet-50 model, to run 90 epochs on imageNet dataset, and to achieve the Top-1 validation accuracy more than 75%. The prerequisite of this evaluation criterion is to achieve the specified accuracy within 90 epochs. We think that there are two shortcomings in this criterion. Firstly, the requirement of 90 epochs limits the real training speed. Due to the influence of convergence rate, the actual training speed and training time are as follows:

$$\text{Actual training speed} = \text{convergence rate} \times \text{single-GPU speed} \times \text{total GPUs} \times \text{GPU scaling efficiency.}$$
$$\text{Actual training time} = 1/\ \text{actual training speed}$$

Accordingly, DOWNBench [22] is used to evaluate the actual training time, which is the training time of the ResNet-50 model when the number of epochs is not limited, and the Top-5 verification accuracy is over 93%.

The second disadvantage is that due to the influence of the number of GPUs, the comparison of training time has become a competition of affordable GPU resources, which does not reflect the advantage of training methods. Some reports compare the GPU scaling efficiency, but this criterion only represents the relative speed of multi-GPU to single-GPU training, and does not represent the absolute speed. Therefore, this paper proposes a new criterion-actual average single-GPU training time: Actual average single-GPU training speed =convergence rate * single-GPU speed * GPU scaling efficiency Actual average single-GPU training time= 1/ actual single-GPU training speed.

The rest of this paper is organized as follows. We will discuss the existing work related to this paper in Section 2.1. The proposed training methods and engineering implementation details of the training system will be discussed in Section 2.2. Section 3 is about the experiments and result analysis. Section 4 concludes this paper.

2. Materials and Methods.

2.1. Related materials.

2.1.1. Gradient descent algorithm. At present, the optimization method for training deep neural network is the first-order optimization algorithm, that is, gradient descent method. The most commonly used one is the stochastic gradient descent method (SGD) [23], which is simple, and has good performance in many applications. However, SGD has a disadvantage, that is, it adjusts the gradient uniformly in all directions. This may cause the convergence rate to change when the training data is sparse. In recent years, many adaptive methods have been proposed to adjust the current gradient direction and magnitude through the direction and magnitude of the historical gradient. These methods include ADAM [24], ADAGRAD [25] and RMSprop [26]. Especially for ADAM, because of its fast convergence rate and stable effect, it is often used in the industry. However, the generalization ability of these adaptive methods is worse than SGD [27]. Recently, Luo et al. proposed a variant of ADAM [20], which is called ADABOUND. By limiting the learning rate in a dynamic upper and lower bound, we can smoothly transform ADAM to SGD. ADABOUND not only retains the fast convergence characteristic of ADAM, but also has the similar generalization ability as SGD.

The general process of gradient descent method is as follows [20]:

1. Calculate the gradient g_t of objective function with respect to current parameter x_t.
2. Calculate current gradient decay: $\eta_t = \alpha_t \cdot m_t/\sqrt{V_t}$.
3. Update parameters: $x_{t+1} = x_t - \alpha_t \cdot m_t/\sqrt{V_t}$.

where, t represents the iterative steps, α_t is the current learning rate of decay. For SGD, $m_t = g_t, V_t = I$. For ADAM, $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$,$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$,$V_t = diag(v_t)$. For ADABOUND, the meanings of the symbols are the same as ADAM, with the value of $\alpha_t \cdot \sqrt{V_t}$limited within the upper and lower boundaries. If $\alpha_t \cdot \sqrt{V_t}$ exceeds the boundaries, it will be truncated, and the upper and lower bounds will be adjusted dynamically. Finally, it will converge to the learning rate α of SGD, to finish the smooth transition from ADABOUND to SGD. β_1 and β_2 are the default hyperparameters.

2.1.2. Distributed training optimization method.

1) Training accuracy optimization. In the era of small batch training, there appears many training tricks for improving the training accuracy, which can still be used for large batch training. Ioffe et al. introduced the BatchNorm technology [21], which normalizes the features in the hidden layers to avoid the gradient vanishing. At present, it has become the standard training module for almost every deep learning framework. The initialization of weights will also affect the final result. Early, Gaussian distribution [1] is used for weight initialization. Later, Xavier initialization [28], KaiMing initialization [29] and other more advanced initialization technologies are proposed. In addition, weight decay is used for weight regularization by $l_1 - norm$, which can improve the generalization ability of the model.

In order to overcome the inherent optimization difficulty [12] and generalization problem of large batch training [17, 18], more specific training skills are required. Goyal et al. proposed an improved version of SGD [12], which has linear scaling on the learning rate. At the same time, the warmup scheme is set up before the linear scaling, that is, a small learning rate is used at the beginning and then it is switched back to the large learning rate when the training process is stable. Google [30] and Sony [15] adopt the strategy of gradually increasing the batch size to ensure the stability of the large batch training process. Because the difference between the weight gradient norms of each layer in neural network leads to the instability of training, [16] proposed a layer-wise adaptive learning rate scaling strategy called LARS, which adjusts learning rate of each layer by multiplying by $\|w\|_2 / \|\nabla w\|_2$, where w is the weight of each layer. LARS achieved excellent results in the super large batch training tasks.

2) Training speed optimization. The precondition of training speed optimization is that there is no loss of the training accuracy, so training speed optimization needs to cooperate with the above training accuracy optimization techniques. On this basis, optimization and improvement can be made in the aspects of single-GPU speed, convergence rate and multi-GPU communication. Akiba et al. [13] finished ResNet-50 training in 15 minutes through RMSprop warmup scheme and the adoption of BN without sliding average. ResNet-50 training has been completed in 6.6 minutes through mixed precision computing, tensor fusion [32], hierarchical and hybrid gradient collective communication by Jia et al. [31]. In addition, [12] and [15] have improved the collective communication part, among which [12] has adopted recursive halving [33] and doubling algorithm [34] and [15] has adopted 2D-Torus all-reduce.

2.2. Methods in this paper.

2.2.1. Hybrid gradient descent. Our goal is to propose a gradient descent method, which can not only ensure fast convergence, but also maintain the advantages of linear scaling SGD in large batch training. Inspired by ADABOUND, we propose an improved version, called linear scaling ADABOUND, which specifies the following restrictions:

– upper bound

$$B_u = lr \cdot N + \frac{1}{(1-\beta_2)\cdot t} - \frac{1}{(1-\beta_2)\cdot T}$$

– lower bound

$$B_l = \frac{t}{T} \cdot lr \cdot N$$

$$\alpha \cdot \sqrt{V_t} = Clip(\alpha \cdot \sqrt{V_t}, B_l, B_u)$$

where, lr represents the learning rate of SGD in single-GPU batch-size. N is the number of GPUs, which meanwhile serves as the linear scaling rate. By comparing with the original ADABOUND, we scale the learning rate to $lr \cdot N$. t is the current epoch. T is the preset ADABOUND maximal epoch. α is the initial learning rate. Clip indicates that the value is limited to the upper and lower bounds, with the exceeding part being truncated. Therefore, when t increases from 0 to T,B_u decreases from $+\infty$ to $lr \cdot N$, while B_l increases from 0 to $lr \cdot N$. When $B_u = +\infty$,$B_l = 0$, the gradient descent method is ADAM; When $B_u = B_l = lr \cdot N$, the gradient descent method is linear scaling SGD. By this, inside each epoch, the smooth transition from ADAM to linear scaling SGD is completed.

In order to further ensure the stability and generalization of the training, we only take the linear scaling ADABOUND as the warmup scheme. After T epochs, the learning rate decay process of SGD starts from the learning rate of $lr \times N$ until the training convergence. In addition to the above, we adopt LARS in SGD stage to adaptively adjust the learning rate of each layer.

To sum up, the complete ADABOUND warmup + linear scaling SGD method is as follows:
For epoch $= 1 : T$

1. Calculate the gradient g_t of the objective function with respect to the current parameter x_t.
2. Average the gradient by sliding:

$$m_t = \beta_1 \cdot m_(t-1) + (1-\beta_1)\cdot g_t$$

3.

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2, \quad V_t = diag(v_t)$$

4. Scale and limit the learning rate to the set range:

$$\alpha/\sqrt{V_t} = Clip(\alpha/\sqrt{V_t}, B_l, B_u)$$

5. Calculate the current gradient decay:

$$\eta_t = \alpha \cdot m_t/\sqrt{V_t}$$

6. Update parameters:

$$x_{t+1} = x_t - \alpha \cdot m_t/\sqrt{V_t}$$

For epoch $= T+1:90$

1. Calculate the gradient g_t of the objective function with respect to the current parameter x_t.
2. Learning rate decreases exponentially:
$$\alpha_t = lr \cdot N \cdot \kappa^{epoch}$$
3. Layer-wise adaptive learning rate scaling:
$$\alpha_t = LARS(\alpha_t)$$
4. Update parameters:
$$x_{t+1} = x_t - \alpha_t \cdot g_t$$

where κ is the default hyperparameter.

The experiments in Section 3 show that compared with using only the ADABOUND or the linear scaling SGD, ADABOUND warmup + linear scaling SGD will produce higher training accuracy.

2.2.2. Gradient computing with mixed precision. The gradient computing with mixed precision scheme in this paper is similar to that in [31]. In the process of forward and back propagation, FP16 precision is used for computing; in the weight updating process, FP32 precision is used for updating. This paper makes the following improvements to this scheme:

1. Since the numerical representation range of FP16 is far smaller than FP32, the conversion from FP32 to FP16 may cause serious optimization problems such as too small weight and gradient being truncated to 0, resulting in gradient vanishing. After the forward propagation is completed, we multiply the loss by a scaling factor, and then continue with backpropagation, which is equivalent to the same scaling when calculating the gradient. This improvement avoids too small gradient value. Meanwhile, it is also easy for implementation.
2. In the process of forward propagation and backpropagation, the operator f with $|f(x)| \gg x$ should be avoided to be calculated under FP16, because the computing results may overflow the representation range of FP16, which makes the numerical accuracy deviate greatly. The specific operators include: exp, square, log and cross entropy loss.
3. In the collective communication process, the gradient of FP16 instead of FP32 is transmitted during gradient aggregation, which reduces the traffic and saves the bandwidth.

2.2.3. Strategy fusion and system engineering implementation. This section lists the problems that need to be noticed in the process of completing the actual distributed training system, as well as the solutions we proposed.

First, based on the improvement proposed earlier in this paper, we adopt a fusion scheme of variety of training strategies.

1) Multi-GPU BN. Compared with the original BN, the following constructions need to be applied in the mean and variance statistics of BN on multi-GPU:

In each iteration, the batch size allocated to each GPU remains unchanged. For neural networks with BN layer, forward loss computing depends on the mean and variance statistics of each single-GPU batch. If the single-GPU batch size changes, the loss function changes, invalidating the premise of linear scaling [12].

In each iteration, the mean and variance of BN are calculated in the batch of each GPU respectively, and the statistics between multiple GPUs are not carried out. On the one hand, the communication bandwidth between multiple GPUs can be saved. On the other hand, the loss computing of each GPU is independent and does not affect each other, meeting the premise assumption of the linear scaling.

The original BN mean and variance calculating method is moving average of the statistical value of each iteration. When the batch is large enough, the computing result of the moving average is relatively inaccurate [13]. Therefore, we only consider the last iteration, and use collective communication to calculate the statistical average over all GPUs. Because large-scale accumulation operation is required, in order to avoid the number value range overflow, we employ FP32 precision.

2) Weight initialization. This paper adopts the following initialization methods:

1. For the convolution layers, the KaiMing Gaussian initialization [28] is applied, that is, the initial weights of convolution layers follow Gaussian distribution $N(0, std)$.

$$std = \sqrt{\frac{2}{fan_{in}}}$$

   where, fan_{in} is the input dimension of the three-dimensional convolution kernel, that is, the length · width · height of the convolution kernel.
2. For the residual block in ResNet, the last layer is BN layer, which performs a linear transformation: $\gamma\widehat{x} + \beta$, where the input of BN layer is denoted by $\widehat{x}$. The scale factor γ of $\gamma\widehat{x} + \beta$ is initialized to 0. Given the input x, assume block(x) is the output of the last layer in the residual block. With combination of the output of the last layer and the input of the residual block, the final output will be block(x)+x. If the scale factor γ of the last BN layer is initialized to 0, then block(x)=0, and only the input is returned finally, which is equivalent to changing ResNet into fewer layers, so that it is easier to train in the initial stage.

3) Multi-machine multi-GPU communication. Because the multi-machine and multi-GPU communication strategy is not the focus of this paper, the collective communication algorithm we adopted is the ring all-reduce [34] based on NCCL. We broadcast from a master node to ensure that the initial weights between multi-machine and multi-GPU are consistent, and distribute data evenly, so as to keep the load balancing of multiple GPUs.

4) Single-GPU computing. In order to improve the GPU scaling efficiency, we let the single GPU complete more computing tasks, and increase the proportion between single-GPU computing time and multi-GPU communication time.

The batch size on a single GPU is made as large as possible until it approaches the upper limit of the GPU memory.

3. Experiments and result analysis.

3.1. Experiment setup.

Hardware. Two training clusters are used in these experiments:

1. Cluster 1 includes 3 machines, with each machine equipped with 8 Nvidia Tesla V100 GPUs with 16G GPU memory, and the GPUs are interconnected through NVlink. 1 NVME 3.2T SSD hard disk is used as the storage. The Connectx-4 Lx EN Cards 40G is employed for the inter-machine communication, with RoCE (RDMA over Converted Ethernet) adopted as the communication mode, which is an RDMA communication mode used under Ethernet, and can be directly connected through memory for quick access to the remote data. We further use GPUDirect RDMA, which allows the direct connection of GPU memory between multiple GPUs to further improve the inter-GPU communication speed.
2. Cluster 2 consists of six machines, each of which is equipped with four Nvidia Tesla P40 GPUs with 24G GPU memory. The GPUs are interconnected by PCIe. One NVME 3.2T SSD hard disk is still employed as the storage. The ConnectX-4 Lx EN Cards 40G is used for inter-machine communication and the communication mode is RoCE. Due to the limitation of GPU type, GPUDirect RDMA cannot be applied in Cluster 2, and the mixed precision cannot be applied neither.

Software. We use horovod [31] as the training framework, which provides APIs to make it easier to write a distributed training script than previous training frameworks such as Distributed TensorFlow. NVIDIA's collective communication library NCCL is used for multi-machine and multi-GPU communication.

Data. The training data is ImageNet database [10], with a total of 1.28 million training pictures and 50000 verification pictures, with the number of categories being 1000. The data augmentation technology in [1] is used in the experiment.

Model. Based on the existing relevant technical reports and the authoritative distributed training test standard DAWNBench [22], ResNet-50 [5] is used as our training neural network model. The basic information of the model is shown in Table 3.1.

Table 3.1
Basic information of ResNet-50 model.

Input Size	Parameter Size	FLOPs	Top-1 Accuracy	Top-5 Accuracy
224*224	25M	4G	75%	93%

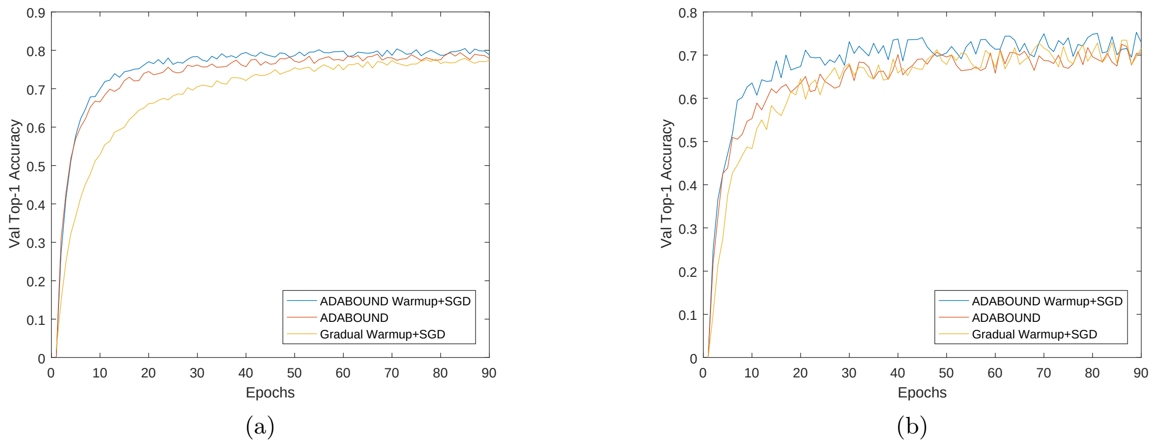

Fig. 3.1. *Top-1 training and validation accuracy curves with different gradient descent algorithms.*

Experimental details. Under the FP32, the batch of size 128 can be loaded on a single V100 GPU, and the total batch-size of cluster 1 is 3072.

Under the mixed precision, we doubled the batch size of V100, because compared with FP32, FP16 only occupies half of the FP32 memory, and the total batch size of cluster 1 is 6144.

Under the FP32, a single P40 GPU can be loaded with batch of size128, and the total batch size of cluster 2 is 3072. Hyperparameters applied in the experiment: the number of iteration epochs of ADABOUND warmup stage is $T = 10$, which is selected according to the experiments shown in section 3.2.1. We directly apply the default hyperparameters for ADABOUND: $\beta_1 = 0.9, \beta_2 = 0.999$. All the other hyperparameters are the same as linear scaling [12]: the single-GPU learning rate $lr = 0.1$. The learning rate of SGD decays exponentially: $lr \cdot \kappa^{epoch}$, where, $\kappa = 0.9$. In SGD, weight decay [1] is adopted, with weight decay is 0.0001.

3.2. Experimental results.

3.2.1. Comparative experiment of gradient descent methods. Listed in this section are the experimental results on cluster 1. The results on cluster 2 are similar to those on cluster 1, therefore will not be discussed again. The gradient descent methods in the comparison include the proposed ADABOUND warmup + SGD, ADABOUND, and gradual warmup + SGD [12] proposed previously. The Top-1 training and validation accuracy curve are shown in Fig. 3.1. This shows that the ADABOUND warmup + SGD method is the most effective, with the convergence rate similar as that of the ADABOUND, and faster than warmup +SGD method. Convergence can be achieved within 50 epochs only. At the same time, on the validation set, the accuracy is the best, Top-1 accuracy = 75.6%, which is about 0.5% higher than that of the second best one, that is, the gradual warmup +SGD method. Because of the fast convergence rate, the accuracy is over 75.0% in the 32nd epoch.

We also tried different epoch T for warmup stopping. The comparison result is shown in Fig. 3.2, which shows that T has little effect on the result. With the increase of T, the convergence rate will be slightly improved, but the training accuracy will also be slightly reduced.

3.2.2. Comparative experiment of gradient computing precisions. Two aspects are compared in this experiment: the training accuracy of mixed precision computing and that of FP32 precision computing, as well as the training speed of mixed precision computing and that of FP32 precision computing. The experi-

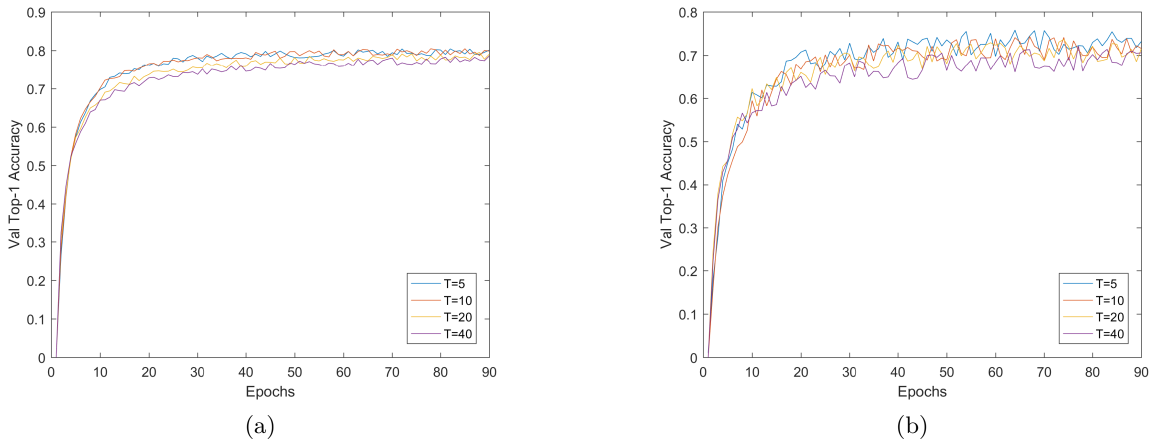

Fig. 3.2. *Top-1 training and validation accuracy curves with different warm-up stopping epoch.*

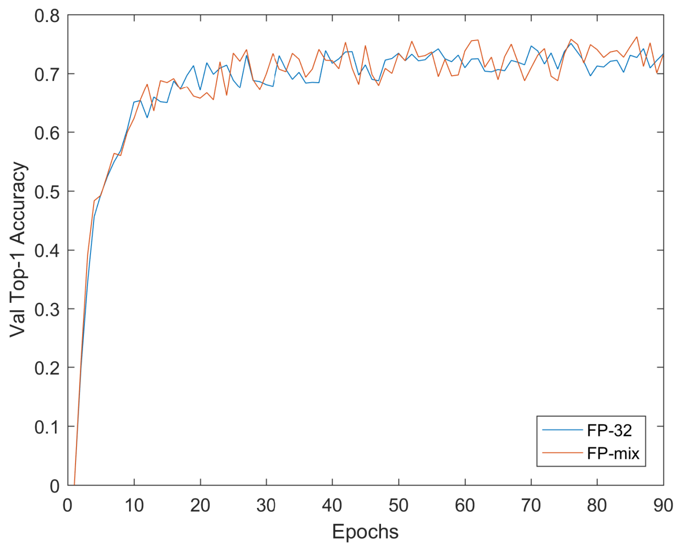

Fig. 3.3. *Top-1 validation accuracy curves with different computing precision.*

ment uses the gradient descent method of ADABOUND warmup + SGD, with other training strategies being consistent with Section 2.2.3.

1) Training accuracy. Because the P40 GPU of cluster 2 does not support FP16 computing perfectly enough, this experiment is only conducted on cluster 1. Figure 3.3 shows the accuracy results of validation set under two precision strategies. The training curve of mixed precision and that of FP32 precision almost coincides, except that the curve of mixed precision is more oscillating. Therefore, by using our methods to avoid the problem of too small numerical range, the mixed precision gradient computing has no effect on the training accuracy.

2) Training speed. First, we test the training speed of single GPU on cluster 1. As shown in Fig. 3.4, the training speed of mixed precision on V100 is much higher than that of FP32. Specifically, when batch size = 128, the training speed of mixed precision is close to twice that of FP32 (Table 3.2). With the maximum batch size 256 under mixed precision, the training speed is 2.03 times faster than that of FP32.

We continue to test the GPU scaling efficiency with mixed precision.

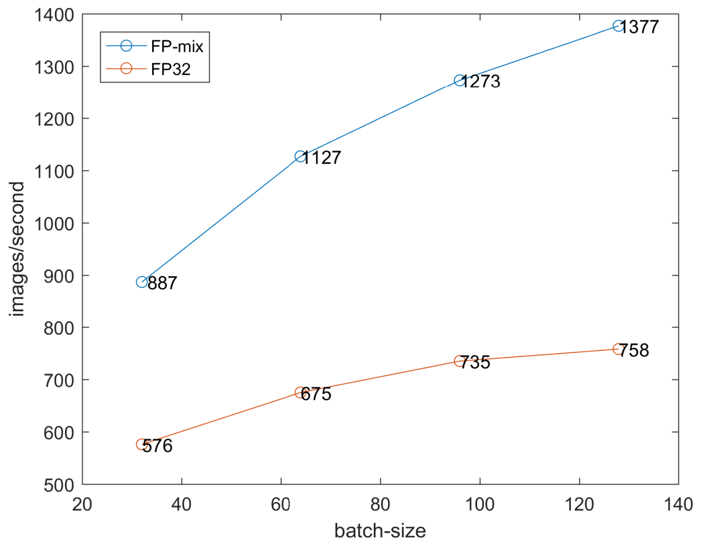

Fig. 3.4. *Training throughput per V100 GPU with different computing precision and batch-size.*

Table 3.2
Training throughput per V100 GPU with different computing precision and batch-size 128/256.

Data Type	Single-GPU Batch-size	Images per Second
FP-mix	128	1377
FP-mix	256	1542
FP32	128	758

The GPU scaling efficiency can be measured by the ratio of actual speed to ideal speed, and can also be measured as the ratio of ideal time-consuming to actual time-consuming. The computing method is as follows:

$$\text{GPU Scaling Efficiency} = \text{Actual Speed/Ideal Speed} = \text{Ideal Time/Actual Time} =$$
$$= 1/(1{+}\text{per GPU Communication Time/per GPU Computing Time}),$$

where

$$\text{ideal time} = \text{per GPU computing time/GPU number},$$
$$\text{actual time} = (\text{per GPU computing time} + \text{communication time})/\text{GPU number}.$$

The computing of mixed precision gradient can reduce the computing time and the communication time of single GPU at the same time. Therefore, when the reduced computing time of single GPU is less than the reduced communication time of single GPU, the scaling efficiency will be improved. On the other hand, when the reduced computing time of single GPU is greater than the reduced communication time of single GPU, the scaling efficiency will be weakened. See Fig 3.5 for the detailed experimental results. When the 24 V100 GPUs of cluster 1 are in full use, the GPU scaling efficiency with mixed precision = actual speed/ideal speed =36082/37008=97.5%. The GPU scaling efficiency of FP32 accuracy = actual speed / ideal speed =17773/18192=97.7%. The mixed precision is slightly lower than that of FP32. However, the speed of single GPU with mixed precision is much faster than that of single GPU with FP32 (see Table 3.2, 1542 images/s:758 images/s), which makes up for the slight deficiency of the GPU scaling efficiency. The total processing speed of 24 GPUs with mixed precision is also much higher than that of FP32, which is 2.03 times (36082 images/s: 17773 images/s). And under different GPU numbers, the linear scaling efficiency is achieved.

In addition, this paper uses the strategy of maximizing the batch size of a single GPU. If the batch size is less than 256 under mixed precision, not only the speed of a single GPU (Table 3.2), but also the GPU scaling efficiency will decrease (Table 3.3), thus affecting the total processing speed of multiple GPUs. Figure 3.6 shows

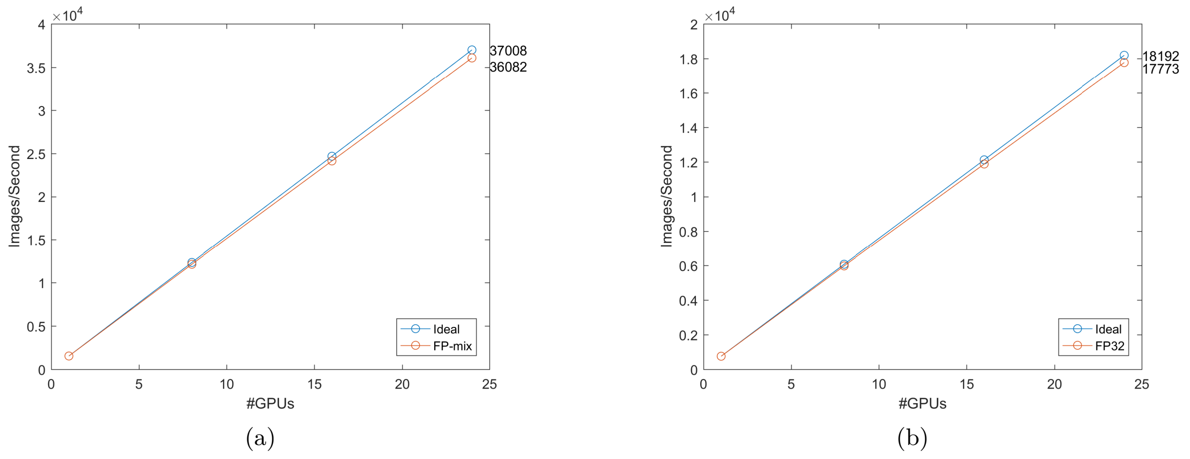

FIG. 3.5. *Multi-V100-GPU training throughput with different computing precision.*

TABLE 3.3
Multi-V100-GPU scaling efficiency with FP16 and different batch-size.

Batch-Size	64	128	192	256
GPU Scaling Efficiency	95.8%	96.4%	97.1%	97.5%

that the total processing speed and the GPU scaling efficiency of 24 GPUs will decrease with the decrease of the batch size.

Next, we test the GPU scaling efficiency improvement only brought by the FP16 collective communication of cluster 2 when the mixed precision computing is disabled. Under the FP32 precision gradient computing, the single-GPU processing speed of P40 is about 295 images/second. As shown in Fig. 3.7, when all 24 P40 GPUs of 6 machines in cluster 2 are in full use, the GPU scaling efficiency of FP16 communication = actual speed/ideal speed =6903/7080=94%. The GPU scaling efficiency of FP32 accuracy communication = actual speed/ideal speed =6456/7080=91.2%. Although the number of GPUs is the same as that of the cluster 1, due to the large number of machines, multi-GPU communication between different machines is more difficult than multi-GPU communication in the same machine, and the communication efficiency is lower. Moreover, it is unable to use GPUDirect RDMA for memory direct connection, so the GPU scaling efficiency of cluster 2 is inherently worse than that of cluster 1. However, through FP16 communication, the GPU scaling efficiency of cluster 2 is greatly improved compared with that of FP32 communication, and the linear scaling is almost achieved under different GPU numbers.

3.2.3. Experiment on the effectiveness of other training strategies. Based on hybrid gradient descent method and the mixed precision computing, we conduct the comparative experiments of the following influencing factors. The experiments are all completed in the cluster 1.

1) Multi-GPU BN. We compare the effects of several BN strategies on training accuracy and training speed. Table 3.4 shows that the Top-1 accuracy is 70.34%, when we randomly change the batch size of single GPU, do BN statistics cross GPUs, and do moving average for each iteration, the Top-1 accuracy is 70.34%.

By using the strategy of this paper, namely, fixed single-GPU batch size, independent BN statistics inside the GPU, and one-time statistics of the last iteration, the Top-1 accuracy can reach 75.6%, which is the best in the experiment.

In addition, cross-GPU BN statistics will have a great impact on communication, and the GPU scaling efficiency will be reduced from the highest value of 97.5% to 96.5%.

2) Weight initialization. We compare three common initialization methods of convolution weight: KaiMing initialization, Xavier initialization and $N(0,1)$ Gaussian initialization. Table 3.5 shows that KaiMing initialization performs best on Top-1 accuracy, in addition, it will improve the convergence rate.

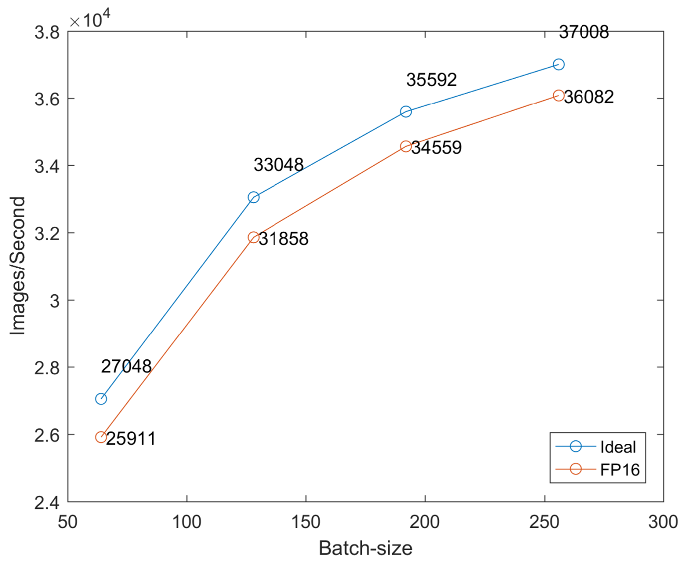

FIG. 3.6. *Multi-V100-GPU training throughput with FP16 and different batch-size.*

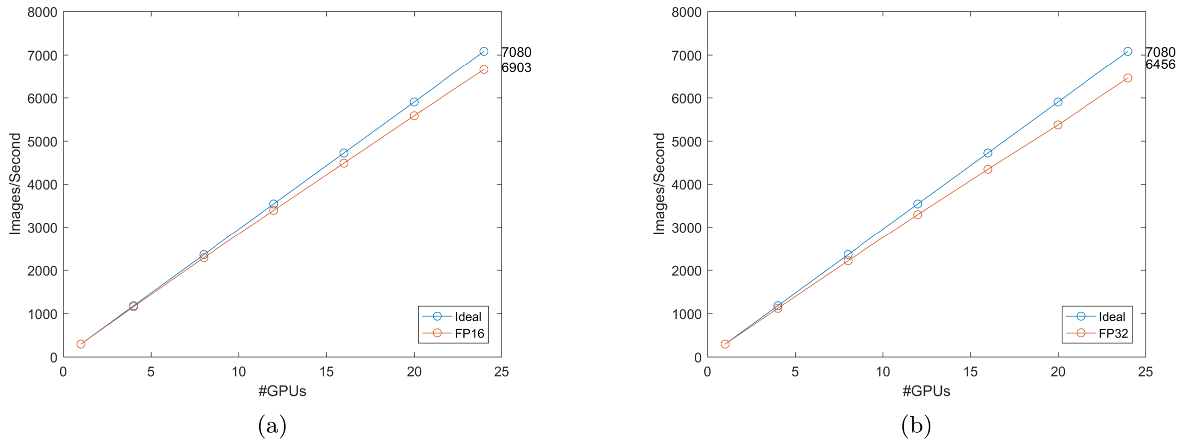

FIG. 3.7. *Multi-P40-GPU training throughput with different collective communication precision.*

TABLE 3.4
Top-1 validation accuracy with various strategies.

Fixed Batch Size	Independent Statistics	Without Moving Average	Top-1 Accuracy
×	×	×	70.34%
√	×	×	73.35%
√	√	×	75.05%
√	√	√	75.60%

TABLE 3.5
Performance with different weight initialization methods.

Initialization for Convolutional Layers	KaiMing Initialization	Xavier Initialization	Gauss Initialization
Top-1 accuracy	75.6%	75.23%	74.38%
Epoch when Top-1 acc=75.0%	32	35	36

Table 3.6
Performance with different γ initialization.

	$\gamma = 0$	$\gamma = 1$
Top-1 accuracy	75.6%	75.32%
Epoch when Top-1 acc=75.0%	32	42

Table 3.7
90-epoch training time and accuracy results for ResNet-50 on ImageNet.

	Time	Top-1 Accuracy	Top-5 Accuracy
Cluster 1	53min	75.6%	93.21%
Cluster 2	108min	75.58%	93.22%

Table 3.8
Actual training time and actual average single-GPU training time for ResNet-50 on ImageNet with Top-1 validation accuracy of 75%

	Actual Training Time	Actual Average Single-GPU Training Time	Convergence Rate	Top-1 Accuracy
Cluster 1	18min	432min	32epoch	>75%
Cluster 2	39min	936min	33epoch	>75%

Table 3.9
Actual training time and actual average single-GPU training time for ResNet-50 on ImageNet with Top-5 validation accuracy of 93%.

	Actual Training Time	Actual Average Single-GPU Training Time	Convergence Rate	Top-5 Accuracy
Cluster 1	23min	552min	39epoch	>93%
Cluster 2	50min	1200min	42epoch	>93%

In addition, we also have conducted comparative experiment on whether to initiate the scale factor γ of the last BN layer of the residual block to 0. The experimental result (Table 3.6) shows that, when γ is initialized to 0, the convergence rate and the training accuracy can both be improved.

3.2.4. Comparison of existing methods. We build a complete distributed training system through the system design of software and hardware integration and high-quality engineering implementation. The absolute time-consuming and final training accuracy of 90 epochs trained on two clusters are listed in Table 3.7 by using the methods and strategies described in Section 2.2.

Taking the convergence rate into account, and by dividing with the number of GPUs, we obtain the actual training time and the actual average single-GPU training time under different accuracy criteria, as shown in Table 3.8 and Table 3.9. Among them, according to the actual training time as the evaluation criterion, our performance under DOWNBench is 23 minutes.

According to the actual average training time of single GPU as the evaluation criterion, the result of cluster 1 can be compared with that of [15, 35] and the result of cluster 2 can be compared with that of [31] since the GPU types are the same. The GPU scaling efficiency of [15, 35, 31] under 24 GPUs has not been published. As the number of GPUs increases, the GPU scaling efficiency generally decreases. For the sake of fairness, we use the published linear scaling efficiency of 99.2% in [31], and set the GPU scaling efficiency of [15, 35] 97.5% as same as ours. From Table 3.10 and Table 3.11, we can see that the actual average single-GPU training time of our method is far less than that of other methods, or even less by more than an order of magnitude. With little difference in the GPU scaling efficiency, the effectiveness of our method is mainly due to fast convergence rate and the super-high acceleration of the single GPU.

4. Conclusion. The main contribution of this paper comes from two aspects: algorithm and system construction. At the algorithm level, this paper proposes a distributed neural network training method based on hybrid gradient computing, which not only improves the convergence rate by hybrid gradient descent while maintaining the generalization ability of SGD, but also improves the speed of single-GPU computing and multi-GPU communication by mixed precision computing. At the system construction level, for achieving

Table 3.10
Actual average single V100-GPU training time with different methods.

	Actual Average Single-GPU Training Time
This work	432min
[15]	5251min
[35]	1941min

Table 3.11
Actual average single P40-GPU training time with different methods.

	Actual Average Single-GPU Training Time
This work	936min
[31]	8908min

excellent training accuracy and speed, we carefully integrate various training strategies and construct a complete distributed training system, which is demonstrated to be effective through a large number of experiments.

In the future, we will continue to improve the training speed and training accuracy. In terms of training speed, the model of ResNet-50, which belongs to the computing intensive model, has a larger ratio of computing volume and parameter volume than other models such as AlexNet and VGGNet, and is easy to achieve higher GPU scaling efficiency. In the future, we will try a variety of all-reduce communication algorithms to achieve 95% or higher GPU scaling efficiency with different types of models (such as IO intensive type). Moreover, we will also try the gradient accumulation on single GPU, which is mainly to increase the upper limit of single GPU's batch size. Before multi-GPU gradient aggregation, we can try to run more batches on the single GPU, and sum the gradients from the backpropagation of each batch cumulatively. In the aspect of training accuracy, we will use the heuristic approach to automatically adjust the hyperparameters that are manually set in the current training process. We believe there will be great possibility to further improve the training accuracy.

Acknowledgments. This paper would not have been possible without the consistent and valuable reference materials that we received from our teachers and friends, whose insightful guidance and enthusiastic encouragement in the course of our shaping this paper gain our deepest gratitude. We shall extend our thanks to editors and specialist reviewer for your instructions and helps to the article.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, *ImageNet classification with deep convolutional neural networks*, NIPS, Lake Tahoe, Nevada, USA, 3-6 Dec 2012.
[2] P. Sermanet, D. Eigen, X. Zhang, et al., *Overfeat: Integrated recognition, localization and detection using convolutional networks*,Arxiv preprint arxiv:1312.6229, 2013.
[3] K. Simonyan, A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, ICLR, Banff, Canada, 14-16 April 2014.
[4] C. Szegedy, W. Liu, Y. Jia, et al., *Going deeper with convolutions*,CVPR, Boston, MA, 7-12 Jun 2015.
[5] K. He, X. Zhang, S. Ren, J. Sun,*Deep residual learning for image recognition*, CVPR, Las Vegas, NV, USA, 27-30 Jun 2016.
[6] G. Hinton, L. Deng, D. Yu, *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, IEEE Signal Processing Magazine 2012, 29(6), 82-97.
[7] W. Xiong, J. Droppo, X. Huang, et al., *The Microsoft 2016 Conversational Speech Recognition System*, ICASSP, Shanghai, China, 21-25 Mar 2017.
[8] R. Collobert, J. Weston, L. Bottou, et al., *Natural language processing (almost) from scratch*, JMLR 2011, 12(1), 2493-2537.
[9] Y. Wu, M. Schuster, Z. Chen, et al., *Google's neural machine translation system: Bridging the gap between human and machine translation*, Arxiv preprint arxiv:1609.08144, 2016.
[10] O. Russakovsky, J. Deng, H. Su, et al., *ImageNet Large Scale Visual Recognition Challenge*, IJCV 2015, 115(3), 211-252.
[11] D. Mahajan, R. Girshick, V. Ramanathan, et al., *Exploring the Limits of Weakly Supervised Pretraining*, ECCV, Munich, Germany, 8-14 Sep 2018.
[12] P. Goyal, P. Dollar, R. Girshick, et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, Arxiv preprint arxiv:1706.02677, 2017.
[13] T. Akiba, S. Suzuki, K. Fukuda, *Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes*, Arxiv preprint arxiv:1711.04325, 2017.

[14] C. Ying, S. Kumar, D. Chen, et al., *Image Classification at Supercomputer Scale*, Arxiv preprint arxiv:1811.06992 v2, 2018.
[15] H. Mikami, H. Suganuma, P. U-chupala, et al., *Massively Distributed SGD: ImageNet/ResNet -50 raining in a Flash*, Arxiv preprint arxiv:1811.05233v2, 2019.
[16] Y. You, I. Gitman, B. Ginsburg, *Large Batch Training Of Convolutional Networkss*, Arxiv preprint arxiv:1708.03888, 2017.
[17] N. S. Keskar, D. Mudigere, J. Nocedal, et al., *On large-batch training for deep learning: Generalization gap and sharp minima*, ICLR, Toulon, France, 24-26 April 2017.
[18] P. Chaudhari, A. Choromanska, S. Soatto, et al., *Entropy-SGD: Biasing Gradient Descent Into Wide Valleys*, ICLR, Toulon, France, 24-26 April 2017.
[19] Y. Yang, Z. Zhang, C. Hsieh, et al.,*ImageNet training in 24 minutes*, Arxiv preprint arxiv:1709.05011, 2017.
[20] L. C. Luo, Y. H. Xiong, Y. Liu, X. Sun, X,*Adaptive Gradient Methods with Dynamic Bound of Learning Rate*, ICLR, New Orleans, LA, USA, 6-9 May 2019.
[21] S. Ioffe, C. Szegedy,*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Arxiv preprint arxiv:1502.03167v3, 2015.
[22] C. A. Coleman, D. Narayanan, D. Kang,*DAWNBench: An End-to-End Deep Learning Benchmark and Competition*, NIPS ML Systems Workshop, Long Beach, USA, 4-9 Dec 2017.
[23] H. Robbins, S. Monro,*A stochastic approximation method*, The Annals of Mathematical Statistics, 1951, 22(3), 400–407.
[24] D. P. Kingma, J. L. Ba,*Adam: A method for stochastic optimization. ICLR, San Diego*, CA, USA, 7-9 May 2015.
[25] J. Duchi, E. Hazan, Y. Singer,*Adaptive subgradient methods for online learning and stochastic optimization*, JMLR, 2011, 12(7), 2121–2159.
[26] T. Tieleman, G. Hinton,*RMSprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, 2012, 4(2), 26–31.
[27] A. C. Wilson, R. Roelofs, M. Stern, et al.,*The marginalvalue of adaptive gradient methods in machine learning*, NIPS, Long Beach, USA, 4-10 Dec 2017.
[28] X. Glorot, Y. Bengio,*Understanding the difficulty of training deep feedforward neural networks*, AISTATS, Chia Laguna Resort, Sardinia, Italy, 13-15 May 2010.
[29] K. He, X. Zhang, S. Ren, et al.,*Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, ICCV, Santiago, Chile, 13-16 Dec 2015.
[30] S. L. Smith, P. J. Kindermans, C. Ying, et al.,*Don't Decay the Learning Rate, Increase the Batch Size*, NIPS, Long Beach, USA, 4-9 Dec 2017.
[31] X. Jia, S. Song, W. He, et al.,*Highly Scalable Deep Learning Training System with Mixed–Precision: Training ImageNet in Four Minutes*, ArXiv preprint arXiv:1807.11205, 2018.
[32] A. Sergeev, M. D. Balso,*Horovod: fast and easy distributed deep learning in TensorFlow*, ArXiv preprint arXiv:1802.05799 (2018).
[33] R. Rabenseifner,*Optimization of collective reduction operations*, ICCS, Krakow, Poland, 6-9 June 2004.
[34] R. Thakur, R. Rabenseifner, W. Gropp,*Optimization of collective communication operations in MPICH*, IJHPCA 2005, 19(1), 49-66.
[35] M. Yamazaki, S. Kasagi, A. Tabuchi,*Yet Another Accelerated SGD: ResNet-50 Training on ImageNet in 74.7 seconds*, ArXiv preprint arXiv:1903.12650.

Edited by: Dana Petcu
Received: Apr 2, 2020
Accepted: May 18, 2020