TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/18 pp114–126
DOI: 10.26599/TST.2020.9010023
Volume 27, Number 1, February 2022

Increasing Momentum-Like Factors: A Method for Reducing Training Errors on Multiple GPUs

Yu Tang, Zhigang Kan, Lujia Yin, Zhiquan Lai, Zhaoning Zhang, Linbo Qiao*, and Dongsheng Li*

Abstract: In distributed training, increasing batch size can improve parallelism, but it can also bring many difficulties to the training process and cause training errors. In this work, we investigate the occurrence of training errors in theory and train ResNet-50 on CIFAR-10 by using Stochastic Gradient Descent (SGD) and Adaptive moment estimation (Adam) while keeping the total batch size in the parameter server constant and lowering the batch size on each Graphics Processing Unit (GPU). A new method that considers momentum to eliminate training errors in distributed training is proposed. We define a Momentum-like Factor (MF) to represent the influence of former gradients on parameter updates in each iteration. Then, we modify the MF values and conduct experiments to explore how different MF values influence the training performance based on SGD, Adam, and Nesterov accelerated gradient. Experimental results reveal that increasing MFs is a reliable method for reducing training errors in distributed training. The analysis of convergent conditions in distributed training with consideration of a large batch size and multiple GPUs is presented in this paper.

Key words: multiple Graphics Processing Units (GPUs); batch size; training error; distributed training; momentum-like factors

1 Introduction

Deep learning has made great progress in numerous fields, such as object detection[2–5], semantic segmentation[6–8], and image classification[9–11]. As the number of layers of neural networks continues to increase and the scale of data continues to expand, the training of deep neural networks has increased the demand for computing power. The development and application of Graphics Processing Units (GPUs) satisfy existing requirements and facilitate the training of deep neural networks within an acceptable time. However, factors, such as GPU memory, still limit large-scale neural network training.

Distributed training architecture provides new ideas for training deep neural networks, including new ideas about distributed training[12–14]. Plenty of paralleled work is proposed recently[15, 16]. Parameter sever[17] is a new type of distributed training architecture, through which we could achieve an ideal reduction of time without compromising accuracy.

Stochastic gradient algorithms, such as Stochastic Gradient Decent (SGD)[18], are commonly used in deep learning and even in certain nonconvex or nonsmooth problems[19–21]. SGD utilizes a random subset of a training dataset to update the weights of the loss function. Batch size can affect training speed and exert nonnegligible influence on the convergence of neural networks. Nowadays, large-scale training has become a research hotspot[22]. Large batch sizes can be used to improve the parallelism of SGD and reduce training

• Yu Tang, Linbo Qiao, Lujia Yin, Zhiquan Lai, Zhaoning Zhang, and Dongsheng Li are with Science and Technology on Paralled and Distributed Processing Laboratory, and College of Computer Science and Technology, National University of Defense Technology, Changsha 473000, China. E-mail: qiao.linbo@nudt.edu.cn; dsli@nudt.edu.cn.

† A part of this paper has been published at Algorithms and Architectures for Parallel Processing, 2020[1].

∗ To whom correspondence should be addressed.

Manuscript received: 2020-06-19; revised: 2020-07-10; accepted: 2020-07-13

time[23–26] while small batch sizes may lead to enhanced training performance at the cost of time.

To fully understand these issues and identify a method to reduce the training errors in distributed training, we train ResNet-50[27] on CIFAR-10[28] and conduct several experiments. In this work, we show how the batch size affects training performance and convergence in a parameter server[17]. We also analyze the influence of the number of GPUs in a parameter server. In our experiments, we fix the total batch size in a parameter server. According to the experimental results, a large batch size and a large number of GPUs adversely influence distributed training performance. To address this problem, we analyze the accuracy drop in distributed training and identify a method related to “momentum” in stochastic gradient algorithms. For convenience, we define a new variable named Momentum-like Factors (MFs), which are factors that control the influence of former gradients on parameter updates in each iteration. Stochastic gradient algorithms are commonly split into first-order algorithms and second-order algorithms and are presented in different ways by using MFs. On the basis of our analysis, we change the MFs in different optimizers, such as SGD, Adaptive moment estimation (Adam), and Nesterov Accelerated Gradient (NAG), and explore how they affect training performance. Moreover, we train a Multilayer Perceptron (MLP) on MNIST and a Long Short-Term Memory model (LSTM) on the Penn TreeBank (PTB) dataset to investigate the influence of MFs. We summarize our main contributions as follows.

- We present our analysis and prove the reason behind the accuracy drop in a multi-GPU training process with Batch Normalization (BN). To the best of our knowledge, this analysis is the first one of its kind in the field of multi-GPU training processes.
- We introduce MFs and explore how they affect training performance in distributed training. We also provide an explanation for the improvement of accuracy after increasing MFs.
- Experiments are conducted on CIFAR-10 by using SGD and Adam, and the influences of a large batch size and different numbers of GPUs are compared. Experiments are also executed on several datasets by modifying the MFs from $\{0.9, 0.95, 0.975, 0.99\}$ on different optimizers. Our experimental results further verify that increasing the MFs can improve the training performance of first-order and second-order stochastic gradient algorithms, especially given large batch sizes and multiple GPUs.

2 Related Work

2.1 Stochastic gradient descent and its variants

SGD[18] is one of the simplest first-order algorithms, but it introduces noise into the gradient and blocks optimization in the training progress[29]. SGD randomly selects one sample or a random sample set at one time for parameter updates so that updating the parameters does not create redundancy. When the data size is large, SGD can effectively accelerate the training process.

Smith and Le[30] stated that SGD should be interpreted as integrating stochastic equations. They also presented the scale of random fluctuations in the SGD dynamics as

$$g = \xi\left(\frac{N}{M} - 1\right) \tag{1}$$

where ξ is the learning rate, N is the size of the training set, and M is the batch size. If we reduce the learning rate ξ, the noise scale g drops, thus leading to an improved training performance. If we keep the learning rate ξ constant, then we could also increase batch size M to reduce the negative impact of noise scale. By contrast, a small batch size increases the noise scale and adversely affects the training performance. According to Ref. [31], calculating the mean and variance values over a batch makes the loss calculated for a particular example dependent on other examples in the same batch. Therefore, if the batch size is large, a high dependency between samples in the batch limits the training performance. This fluctuation scale g can also be considered as a *noise factor*. When $M \ll N$, applying a linear scaling rule[32] maintains the mean SGD weight update constant per training sample. A specific description about the linear scaling rule[32] is shown in Section 2.2.

SGD has several variants[33–35]. A common one is SGD with momentum[36]; specifically, Smith and Le[30] extended the traditional SGD to include momentum, and found that the noise factor g changes into

$$g = \frac{\xi}{1-m}\left(\frac{N}{M} - 1\right) \approx \frac{\xi N}{M(1-m)} \tag{2}$$

where m is the momentum. Equation (2) degenerates into Eq. (1) when $m = 0$. If a linear scaling rule is adopted, ξ/M is constant. Then, we obtain $g \propto 1/(1-m)$. Increasing m results as g rises may cause a drop in generalization performance. Adam[37] combines the advantages of two optimization algorithms, namely, AdaGrad[38] and RMSProp[39]. Adam evaluates the first moment estimation of a gradient and the second moment estimation and then calculates the update step.

This algorithm is a second-order optimization method, which adjusts the learning rate for each parameter by performing small updates for frequently used parameters and large updates for seldomly used parameters. In the vanilla Adam algorithm[37], β_1 and β_2 are set to control the influence of gradients and the square of gradients on the parameter update, respectively. They play a similar role to momentum in the SGD method with momentum. NAG[40] is an improved method of momentum[36] and a first-order optimizer. It updates with the gradient by "looking ahead" instead of using the current gradient. Moreover, it calculates the variance of gradients with respect to the last one. By utilizing these values, NAG updates the parameters in the training process.

SGD and its variants, known as stochastic gradient algorithms, can be regarded as first-order or second-order stochastic gradient algorithms, as shown in Algorithms 1 and 2, respectively.

Algorithm 1 First-order stochastic gradient algorithms with MF

```
Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\xi_t\}_{t=1}^T$ (we use a constant $\xi$ in our experiments), momentum-like factor MF, batch size $M$, and optimizer OP.
Output: $\theta_T$
1: Init MF and $v_0 = 0$
2: for $t = 1$ to $T$ do
3:     Draw $M$ samples $\mathcal{B}_t$ from $\mathcal{B}$
4:     Compute $g_t \leftarrow \frac{1}{M} \sum_{x_t \in \mathcal{B}_t} \nabla L_t(\theta_t, x_t)$
5:     $v_t \leftarrow \mathrm{MF} \cdot v_{t-1} + \xi g_t = \mathrm{MF} \cdot v_{t-1} + G \cdot \sum_{x_t \in \mathcal{B}_t} \nabla L_t(\theta_t, x_t)$
6:     $\theta_t \leftarrow \theta_{t-1} - v_t$
7: return $\theta_T$
```

Algorithm 2 Second-order stochastic gradient algorithms with MF

```
Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\xi_t\}_{t=1}^T$ (we use a constant $\xi$ in our experiments), momentum-like factors MF$_1$ and MF$_2$, batch size $M$, optimizer OP, and scaling function $\phi$.
Output: $\theta_T$
1: Init MF$_1$, MF$_2$, $m_0 = 0$, and $v_0 = 0$.
2: for $t = 1$ to $T$ do
3:     Draw $M$ samples $\mathcal{B}_t$ from $\mathcal{B}$
4:     Compute $g_t \leftarrow \frac{1}{M} \sum_{x_t \in \mathcal{B}_t} \nabla L_t(\theta_t, x_t)$
5:     $m_t \leftarrow \mathrm{MF}_1 \cdot m_{t-1} + (1 - \mathrm{MF}_1) \cdot g_t$
6:     $v_t \leftarrow \mathrm{MF}_2 \cdot v_{t-1} + (1 - \mathrm{MF}_2) \cdot g_t^2$
7:     $\theta_t \leftarrow \theta_{t-1} - \xi \cdot \phi(m_t, v_t, \mathrm{MF}_1, \mathrm{MF}_2)$
8: return $\theta_T$
```

2.2 Linear scaling rule

Assume a deep neural network model with parameters θ, and its corresponding loss function $L(\theta)$. Thus, $L(\theta)$ is defined as the average of the total loss over the training dataset. The formula is as follows:

$$L(\theta) = \frac{1}{M} \sum_{i=1}^{M} L_i(\theta),$$

where $L_i(\theta)$ is the loss of the i-th training example. As stated in Section 2.1, SGD uses one stochastic sample or one stochastic sample set to obtain the approximation of the gradient of the loss function $L(\theta)$. For batch $\mathcal{B}$ which includes m training examples, the batch size is m. Its corresponding weight update rule is

$$\theta_{k+1} = \theta_k + \xi \Delta\theta_k,$$
$$\xi \Delta\theta_k = -\frac{\xi}{m} \sum_{i=1}^{m} \nabla_\theta L_i(\theta_k) \tag{3}$$

where k is the k-th epoch.

From Eq. (3), we can obtain $\mathcal{E}\{\xi\Delta\theta\} = -\xi\mathcal{E}\{\nabla_\theta L(\theta)\}$, which denotes the average weight update of SGD could be kept when changing the learning rate proportionally. Thus, for batch $\mathcal{B}$ whose batch size is m, the mean value of the SGD weight update is proportional to ξ/m. Adopting the *linear scaling rule* involves keeping the mean SGD weight update per training example constant[31].

This linear scaling rule is adopted widely in Refs. [30, 32, 41–43]. We apply the linear scaling rule to our theoretical analysis of Algorithms 1 and 2 (Section 3.2). This rule is an important foundation of our algorithms and experiments.

2.3 Batch normalization

BN[44] is commonly deployed in modern deep neural networks. It has shown excellent achievements in improving training performance. However, it also causes performance decline in multi-GPU distributed training. The standard BN is shown as follows.

For batch $\mathcal{B} = \{x_0, x_1, \ldots, x_{M-1}\}$ with batch size M, its mean value is

$$\mu_{\mathcal{B}} = \frac{1}{M} \sum_{i=0}^{M-1} x_i,$$

where x_i is one sample of batch $\mathcal{B}$. Its variance is

$$\delta_{\mathcal{B}}^2 = \frac{1}{M} \sum_{i=0}^{M-1} (x_i - \mu_{\mathcal{B}})^2 \tag{4}$$

Then, the samples are normalized by

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\delta_{\mathcal{B}}^2 + \epsilon}} \tag{5}$$

where i changes from 1 to M and ϵ is a small enough to avoid a zero denominator for Eq. (5). The mathematical expectation is

$$\mathcal{E}_1 = E(\delta_{\mathcal{B}}^2) = (M-1)\delta^2 \tag{6}$$

After BN, the samples follow the normal distribution whose variance is δ^2. These values are fed into several layers in the deep neural network to achieve superior training results. This analysis is important in multi-GPU distributed training, as described in Section 3.1.

3 Algorithm

3.1 Multi-GPU training

As discussed in Section 2.3, the standard BN is analyzed in this work. At this point, we consider the training on multiple GPUs and explore how it affects the performance with BN. Assume P GPUs in the distributed training system and batch $\mathcal{B} = \{x_0, x_1, \ldots, x_{M-1}\}$ whose batch size is M. The batch per GPU is a subset of batch $\mathcal{B}$, and its batch size is $K = M/P$. On GPU $j (j = 0, \ldots, P-1)$, the mean value of its training samples is

$$\mu_j = \frac{1}{K}\sum_{i=0}^{K-1} x_{ji} = \frac{P}{M}\sum_{i=0}^{M/P-1} x_{ji},$$

and its corresponding variance is

$$\delta_j^2 = \frac{1}{K}\sum_{i=0}^{K-1}(x_{ji}-\mu_j)^2 = \frac{P}{M}\sum_{i=0}^{M/P-1}(x_{ji}-\mu_j)^2,$$

where x_{ji} is the i-th training sample on GPU j. The mean value of all samples of batch $\mathcal{B}$ is

$$\hat{\mu}_{\mathcal{B}} = \frac{1}{P}\sum_{j=0}^{P-1}\mu_j.$$

The sum of each variance of all GPUs in the system is

$$S_e = \sum_{j=0}^{P-1}\frac{M}{P}\delta_j^2 = \frac{M}{P}\sum_{j=0}^{P-1}\delta_j^2.$$

The variance of all training batches among GPUs is

$$S_a = \sum_{j=0}^{P-1}\frac{M}{P}(\mu_j - \hat{\mu}_{\mathcal{B}})^2 = \frac{M}{P}\sum_{j=0}^{P-1}(\mu_j - \hat{\mu}_{\mathcal{B}})^2.$$

Thus, the total variance S_t is

$$S_t = S_e + S_a = \frac{M}{P}\sum_{j=0}^{P-1}\delta_j^2 + \frac{M}{P}\sum_{j=0}^{P-1}(\mu_j - \hat{\mu}_{\mathcal{B}})^2 \tag{7}$$

When $P=1$, Eq. (7) degenerates into Eq. (4). According to statistical knowledge, the expectation of Eq. (7) is

$$\mathcal{E}_2 = E(S_t) = (M-1)\delta^2 + \frac{M(P-1)}{P}\delta^2 \tag{8}$$

By comparing Eq. (6) with Eq. (8), we obtain $\mathcal{E}_2 \geqslant \mathcal{E}_1$ while ignoring the dependencies between training samples if $P \geqslant 1$, in which case the input features after BN decline in multi-GPU training. Therefore, the variance of training on multiple GPUs results in performance degradation in distributed training.

3.2 Momentum-like factor

As the variance of BN increases, the gradients calculated on each worker in the parameter server tends to be unstable. The gradients in multi-GPU training are more updated than those on a single GPU. Herein, we explore a new method to reduce the training errors on multiple GPUs. We define the MF as a new variable that controls the influence of the last gradient on the current parameter update in each iteration. We formalize the stochastic gradient algorithms, including the first-order and second-order stochastic gradient algorithms, with MFs. The two types of algorithms are abstracted into Algorithms 1 and 2. We apply the linear scaling rule[32] to the two algorithm types.

In general, the set of learning rate $\{\xi_t\}_{t=1}^T$ in Algorithms 1 and 2 could contain more than one value. However, we keep the learning rate constant in each training process. Thus, the set of learning rate $\{\xi_t\}_{t=1}^T$ changes into a constant ξ. As we apply the linear scaling rule, $\dfrac{\xi}{M} = G$ remains unchanged in each iteration.

Let us consider training on multiple GPUs with MF using first-order stochastic gradient algorithms with MF (Algorithm 1). The MF update rule for the proportion of the former gradient in the training process[45] is

$$\Delta A = -(1-\mathrm{MF})A + \frac{\mathrm{d}L(\theta)}{\mathrm{d}w} \tag{9}$$

$$\Delta w = -A\xi \tag{10}$$

where A denotes the "accumulation" and ΔA is its variety while $\dfrac{\mathrm{d}L(\theta)}{\mathrm{d}w}$ is the average gradient of per training example. Increasing the MF can lower ΔA and slow down the attenuation of weights. Therefore, we obtain Δw, which is relatively small. This property correspondingly reduces the training errors on multiple GPUs in the same iteration.

Let us focus on Algorithm 1. For Line 5 in

Algorithm 1, let us assume $\sum_{x_t \in \mathcal{B}_t} \nabla L_t(\theta_t, x_t) = \sum_{x_{t-1} \in \mathcal{B}_{t-1}} \nabla L_{t-1}(\theta_{t-1}, x_{t-1})$, that is, $\nabla L(\theta, x)$ is constant in each iteration. Therefore, increasing the MF in Algorithm 1 results in an increase in v_t. The outcome leads to a small parameter update, which is the same as that achieved in the previous analysis and leads to good training performance.

As for the second-order stochastic gradient algorithms with MF_1 and MF_2 (Algorithm 2), the "accumulation" update is

$$\Delta A = -(2 - \text{MF}_1 - \text{MF}_2)A + \frac{\mathrm{d}L(\theta)}{\mathrm{d}w} \tag{11}$$

Then, the parameter update follows Eq. (10). In Algorithm 2, ϕ is a function of m_t, v_t, MF_1, and MF_2. For example, in Adam, the function could be $\phi(m_t, v_t, \text{MF}_1, \text{MF}_2) = \dfrac{m_t\sqrt{1-\text{MF}_2}}{(\sqrt{v_t}+\epsilon)(1-\text{MF}_1)}$, where ϵ is a relatively small value, such as 10^{-8}, to avoid a zero denominator.

By applying a linear scaling rule[32] to Algorithm 2, we obtain a constant G, which is similar to the former one. Hence, we mainly focus on $\xi \cdot \phi(m_t, v_t, \text{MF}_1, \text{MF}_2)$ in Line 7 (Algorithm 2). The function ϕ plays an important role in this analysis. Take Adam as an example, $\phi(m_t, v_t, \text{MF}_1, \text{MF}_2) = \dfrac{m_t\sqrt{1-\text{MF}_2}}{(\sqrt{v_t}+\epsilon)(1-\text{MF}_1)}$. If we only change MF_1 rather than MF_2, we can view all formulas with MF_2 and v_t as constants G_1 and G_2, respectively.

According to our analysis, increasing MF_1 in Algorithm 2 has the same effect as increasing the MF in Algorithm 2. The detailed analysis is shown in Appendix A.

For ease of study, we present the influence of increasing the MF value in stochastic gradient algorithms in Figs. 1 and 2. With an increase in the number of iterations, the training process tends to be stable, and

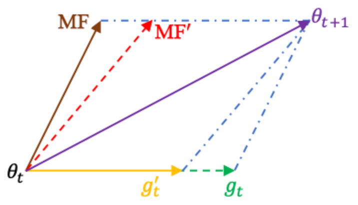


Fig. 1 Parameter update change of increasing the MF value when g_t declines. If we want to obtain the same θ_{t+1} given the decline of gradients, namely, $g_t' < g_t$, we ought to increase from the MF (the brown line) to the MF′ (the red line). This method reduces the training errors accordingly.

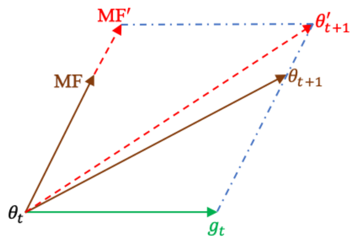


Fig. 2 Parameter update change to increase the MF value on the condition that the gradients do not change. Increasing from MF to MF′ results in a new parameter update direction, θ'_{t+1} (the red one). This new direction could correct the error caused by a large batch size or multi-GPU training.

the gradients in each iteration tend to decline. Therefore, the change in gradients in the training process declines either minimally, with the change almost left unseen, or excessively. As shown in Fig. 1, θ_t is the parameter in the t-th iteration. Taking this situation as an example and for the sake of simplifying, we use MF (the brown line) to represent the updates resulting from the MF value in the training process. The same is true for MF′ (the red line). In Fig. 1, g_t (the green line), (resp. g'_t, the yellow line) marks the gradients calculated from the samples in the t-th iteration with respect to MF (resp. MF′). In large-scale distributed training, we may obtain $g'_t < g_t$ as a result of a large batch size or multi-GPU training. Therefore, if we want to obtain the same θ_{t+1}, the parameter in the $(t+1)$-th iteration (the purple line, then we should change MF to MF′. In this situation, we have MF′ > MF. Therefore, increasing the MF value could alleviate the drop in accuracy in distributed training to some extent, that is, the training errors in distributed training could be reduced, especially when the batch size is large and when multiple GPUs are used in the training process. In certain situations, we may achieve a good parameter update (θ_{t+1}) as a result of an increase in MF values. Considering the slight decline of the gradients, we arrive at the situation shown in Fig. 2. The variables have the same meaning as those in Fig. 1. In this situation, we might obtain a poor parameter update (θ_{t+1}) as a result of the large-scale training or training on multiple GPUs. Therefore, increasing the MF value corrects the parameter update (θ_{t+1}) and reduces the training errors correspondingly.

3.3 Distributed algorithm

In this section, we present distributed algorithms. As we use the parameter server architecture in our experiments, we show the training algorithms of *workers* and *servers* in Algorithms 3 and 4, respectively.

Algorithm 3 refers to the training algorithms on workers in the parameter server. In Algorithm 3, we input all the hyperparameters needed by the experiments and allocate P GPUs as *workers* in the initialization phase. In the t-th iteration, the workers give the servers the signals of *Pull* trigger and *Pull* last saved parameter (θ_{t-1}). These parameters are fed into Algorithms 1 or 2 according to OP. Finally, workers send the *Push* trigger and *Push* θ_t to the servers. Algorithm 4 shows the training algorithms on servers in the parameter server architecture. It has the same initialization phase as Algorithm 3 but a different process. In the t-th iteration, when receiving the *Pull* trigger from the workers, the servers *Push* θ_{t-1} to the workers. After receiving the signal of *Push* trigger, the servers are responsible for *Pull* θ_t and save it locally.

Algorithms 1 and 2 are performed on the basis of the first- and second-order stochastic gradient. All of our

Algorithm 3 Training algorithm-workers

```
Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\xi$, momentum-like factors MF for first-order stochastic gradient algorithms (MF$_1$ and MF$_2$ for second-order stochastic gradient algorithms), the total batch size $M$, optimizer OP (such as SGD), the number of GPUs $P$, iteration number $T$, and the number of nodes $N$ (the scaling function $\phi$ for second-order stochastic gradient algorithms).
Output: $\theta_T$, Pull trigger, and Push trigger.
1: Init: allocate $P$ GPUs as workers.
2: for $t = 1$ to $T$ do
3:     Send Pull trigger to servers.
4:     Pull $\theta_{t-1}$ from servers.
5:     if OP is a first-order stochastic gradient algorithm then
6:         Input $\theta_{t-1}$, $\xi$, MF, $M/N$, OP into Algorithm 1.
7:         Get the output $\theta_t$ of Algorithm 1.
8:     if OP is a second-order stochastic gradient algorithm then
9:         Input $\theta_{t-1}$, $\xi$, MF$_1$, MF$_2$, $M/N$, $OP$ into Algorithm 2.
10:        Get the output $\theta_t$ of Algorithm 2.
11:    Send Push trigger to servers.
12:    Push $\theta_t$ to servers.
13: return $\theta_T$
```

Algorithm 4 Training algorithm-servers

```
Input: the iteration number $T$, Pull trigger, and Push trigger.
1: Init: allocate $P$ GPUs as servers.
2: for $t = 1$ to $T$ do
3:     if Pull trigger then
4:         Push $\theta_{t-1}$ to workers.
5:     if Push triggers then
6:         Pull $\theta_t$ and save.
7: return
```

experiments are based on these algorithms.

4 Experiment and Analysis

To explore the impact of batch size and the number of GPUs on the performance of the algorithms in the parameter server architecture, we conduct experiments by using MXNet[46] and train ResNet-50[27] on CIFAR-10. We are aware that training performance may decline as the batch size and number of GPUs increase. To eliminate the negative impact of a large batch size and multiple GPUs, we increase the MFs in different stochastic gradient algorithms.

4.1 Experimental setup

4.1.1 Dataset

Experiments are executed on three different datasets, namely, CIFAR-10[28], MNIST[47], and Penn Treebank (PTB)[48].

The CIFAR-10 dataset consists of 60 000 natural images in 10 classes of 32×32 resolution on Red, Green, and Blue (RGB) scale; 50 000 of these images are training images, and 10 000 are test images[28]. MNIST has 55 000 training samples and 10 000 test samples and was first introduced in Ref. [47] to train digital handwritten image recognition. The PTB dataset comprises 923 000 training, 73 000 validation, and 82 000 test words[48], and it is a commonly used dataset in the field of language models.

4.1.2 Setup

In our experiments, we train ResNet-50 on CIFAR-10 by adopting three optimizers, namely, SGD, Adam, and NAG. We then set the batch sizes to 32, 64, 128, 256, 512, 1024, and 2048. The experiments are conducted across multiple GPUs simultaneously by using 1, 2, 4, 8, 16, and 32 GPUs to evaluate the validation accuracy of ResNet-50 on CIFAR-10 in the case of different batch sizes and GPU numbers in the parameter server architecture. The GPUs are Tesla K80. In these experiments, we train 300 epochs and set $\text{MF} = 0.9$ in SGD and NAG, and $\text{MF}_1 = 0.9, \text{MF}_2 = 0.999$ in Adam. We also set a default value of batch size $= 128$ and learning rate $= 0.05$. We also adopt a linear scaling rule[32] in our experiments as well. The learning rates corresponding to different batch sizes are displayed in Table 1.

Inspired by Ref. [49], we focus on setting the MFs to 0.9, 0.95, 0.975, and 0.99 in the first-order stochastic gradient algorithms (MF in Algorithm 1)

Table 1 Learning rates corresponding to different batch sizes.

Batch size	Learning rate	Batch size	Learning rate
32	0.0125	512	0.2
64	0.025	1024	0.4
128	0.05	2048	0.8
256	0.1		

and second-order stochastic gradient algorithms (MF_1 in Algorithm 2). We then analyze how they influence training performance given a large batch size of 1024. In addition, we also train an MLP on MNIST[47]. For the sake of convenience, we use "momentum" to represent MF in the corresponding optimizer.

4.2 Training ResNet-50 on CIFAR-10

We use two different optimizers, namely, SGD and Adam, in training ResNet-50 on CIFAR-10. MF is set to 0.9 in SGD. As for the Adam optimizer, MF_1 and MF_2 are set to 0.9 and 0.999, respectively. Figures 3 and 4 show the results of SGD and Adam, respectively. For convenience, we omit in Fig. 3 the validation accuracy of 0.199 given a batch size of 32 and 32 GPUs. These values are also shown in Table 2. As for Adam, the results of the training on multiple GPUs in the parameter sever are summarized in Table 3.

4.2.1 Batch size

In distributed training, the choice of batch size is critical. According to the curves in Figs. 3 and 4, a large batch size causes the validation accuracy to drop given a constant number of GPUs. As shown in Fig. 3, when the batch size is 32 and the number of GPUs is 32, the validation accuracy is 0.199, which means that in distributed training, such a small batch size cannot make the training process converge. In this case, the batch size on each node is four. The batch size is too small for the training process to converge. From Fig. 3, we observe a downtrend when the batch size increases. When the batch size exceeds 512, the effect of increasing the batch size on the experimental results is small, and the maximum drop is 0.008 (16 GPUs). Nearly the same situation happens in Fig. 4. The setting causes a bad validation performance under the condition with 32

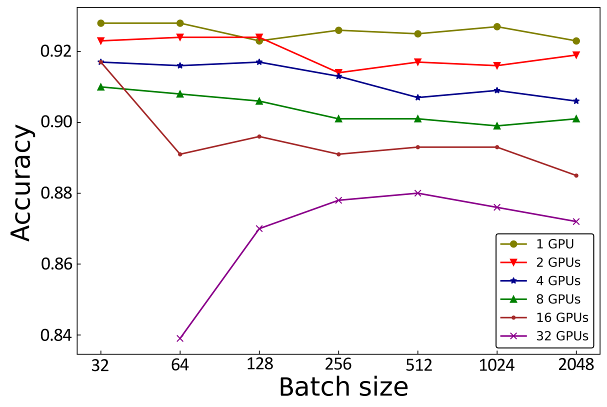


Fig. 3 Validation accuracy of Resnet50 on CIFAR-10 of different batch sizes on multiple GPUs in the parameter server based on SGD. In these experiments, all hyperparameters, except batch size and number of GPUs, are set to their default values.

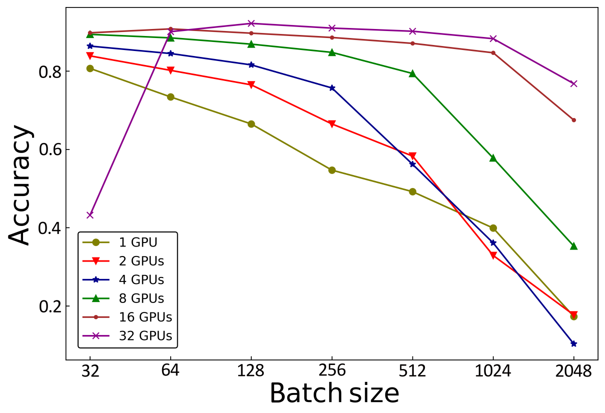


Fig. 4 Validation accuracy of Resnet50 on CIFAR-10 of different batch sizes on multiple GPUs in the parameter server based on Adam. The hyperparameters are set to be the same as those in the previous SGD experiments.

Table 2 SGD's validation accuracy for ResNet-50 on CIFAR-10 of different batch sizes on multiple GPUs in the parameter server.

Batch size	Validation accuracy					
	1 GPU	2 GPUs	4 GPUs	8 GPUs	16 GPUs	32 GPUs
32	0.928	0.923	0.917	0.910	0.917	0.199
64	0.928	0.924	0.916	0.908	0.891	0.839
128	0.923	0.924	0.917	0.906	0.896	0.870
256	0.926	0.914	0.913	0.901	0.891	0.878
512	0.925	0.917	0.907	0.901	0.893	0.880
1024	0.927	0.916	0.909	0.899	0.893	0.876
2048	0.923	0.919	0.906	0.901	0.885	0.872

Table 3 Adam's validation accuracy for ResNet-50 on CIFAR-10 of different batch sizes on multiple GPUs in the parameter server.

Batch size	Validation accuracy					
	1 GPU	2 GPUs	4 GPUs	8 GPUs	16 GPUs	32 GPUs
32	0.807	0.839	0.864	0.894	0.898	0.432
64	0.734	0.802	0.845	0.885	0.908	0.901
128	0.665	0.765	0.816	0.869	0.897	0.922
256	0.547	0.665	0.757	0.848	0.886	0.910
512	0.492	0.583	0.562	0.794	0.871	0.902
1024	0.399	0.329	0.361	0.578	0.847	0.883
2048	0.173	0.177	0.103	0.353	0.675	0.768

GPUs and a batch size of 32. Thus, we conclude that in multi-GPU situations, a large batch size may increase parallelism in distributed training, but the validation accuracy tends to drop.

4.2.2 GPU

When the numbers of GPUs are 1, 2, and 4, we only need to use a single node. According to Figs. 3 and 4, when the GPU number changes from 1 to 4 within a single host, the accuracy difference is relatively small. In Fig. 3, the largest accuracy difference of 0.018 lies in batch sizes 512 and 1024. When using multiple GPUs, especially 32 GPUs, the accuracy drops by 0.069 (batch size of 64) given a convergent training process. Nearly for each line in Figs. 3 and 4, the validation accuracy declines if the number of GPUs increases for a constant batch size.

4.3 SGD versus Adam

4.3.1 Experimental result

In this section, we mainly focus on the influence of different MF values on SGD and Adam. As discussed previously, increasing the number of GPUs in the parameter server causes a decline in training accuracy when the batch size remains constant. To improve the accuracy in this situation, we increase the MFs and compare the improvements caused by different MFs given a large batch size of 1024 and 16 GPUs.

We train ResNet-50 on CIFAR-10 by utilizing the optimizers SGD and Adam. The batch size is 1024, and the learning rate is set to 0.4 (Table 1). The validation accuracy of SGD is illustrated in Fig. 5. When the batch size is 1024 and the momentum is 0.9, the final validation accuracy is 0.927 on 1 GPU and drops to 0.876 on 32 GPUs (green line). When the momentum is 0.95, the validation accuracies are 0.93 and 0.892 on 1 GPU and 32 GPUs, respectively (red line). Relative to those when the momentum is 0.9, these accuracy values increase by 0.003 and 0.016. When the momentum increases to 0.99, the accuracies are 0.904 on 1 GPU and 0.907 on 32 GPUs (purple line). Compared with that indicated by the green line (momentum is 0.9), the accuracy drops by 0.026 on 1 GPU but increases by 0.031 on 32 GPUs. For SGD, we find that increasing the MF can effectively alleviate the accuracy degradation caused by the increase in the number of GPUs in the parameter server. The improvement is particularly obvious for multi-GPU scenarios. Therefore, increasing the MFs yields better effects in the first-order stochastic gradient algorithms than in the second-order ones.

As far as Adam, we change MF_1 from 0.9 to 0.95, 0.975, and 0.99 to explore the influence of such change on distributed training given the same batch size and learning rate. We display the validation accuracy of Adam in Fig. 6. We observe that increasing MF_1 can improve distributed training performance. The improvements are not obvious in Adam. Moreover, we spot trends that are not similar to those in Fig. 6. A decline in accuracy occurs when $\mathrm{MF}_1 = 0.9$, 0.95, and 0.975 when the number of GPUs increases from 1 to 4. When MF_1 is 0.99, such decline does not occur. Increasing MF_1 still alleviates the negative impact of multiple GPUs on accuracy and stabilizes the training process on multiple GPUs in Adam. Therefore, the influence of increasing MFs in second-order stochastic gradient algorithms is not as obvious as that in first-order stochastic gradient algorithms. Nevertheless, increasing MFs remains effective and leads to a relatively stable trend when the number of GPUs increases.

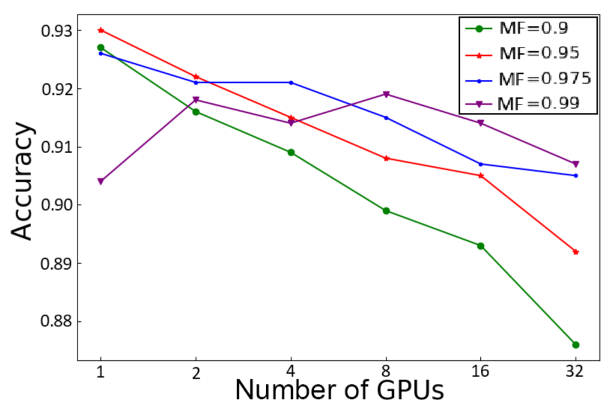


Fig. 5 Validation accuracy of different MFs on multiple GPUs utilizing SGD. In these experiments, we set the MF values 0.9, 0.95, 0.975, and 0.99, respectively.

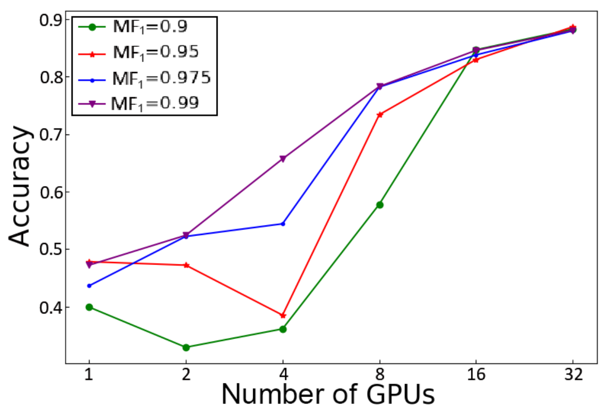


Fig. 6 Validation accuracy of different MF_1s on multiple GPUs utilizing Adam. In these experiments, we set the MF_1 values 0.9, 0.95, 0.975 and 0.99, respectively.

4.3.2 Convergent condition

In certain situations in Section 4.3.1, the results show that the training process does not converge. In this section, we provide our convergence analysis and establish a number of convergent conditions in which MFs are considered in distributed training.

As pointed out previously, we view the deep learning problem as

$$\min_{\theta\in\mathbb{R}^d} G(\theta) = \mathcal{E}[f(\theta)] = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta) \tag{12}$$

where $f(\theta)$ is a nonconvex loss function and each $f_i : \mathbb{R}^d \to \mathbb{R}$ is a smooth and nonconvex function.

For the first- and second-order stochastic gradient algorithms, we have the following assumptions.

- A1: The gradient of f is L-Lipschitz continuous, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \leqslant L\|x-y\|, \forall x, y \in \mathbb{R}^d$. Then we have $G(x) \leqslant G(y) + \langle\nabla G(y), x-y\rangle + \frac{L}{2}\|x-y\|^2$;
- A2: The minimum value of Eq. (12) is lower-bounded, i.e., $G^* = \min\limits_{\theta\in\mathbb{R}^d} f(\theta) > -\infty$;
- A3: The gradient g_t of $f_t(\theta_t)$ is an unbiased estimate;
- A4: The gradients are bounded, i.e., $\nabla f_t \leqslant C$;
- A5: The second-order moment of the gradient g_t is uniformly upper-bounded, that is to say, $\mathcal{E}\|g_t\|^2 \leqslant C$.

In A4 and A5, $C < +\infty$ and it is a constant. Such assumptions are typical in the analysis of stochastic gradient methods[50, 51].

From A3 and A4, in each iteration, we have $\frac{1}{M_{\text{node}}}\sum\limits_{\theta\in\mathbb{R}^d} f_t(\theta_t) \leqslant C$, where M_{node} is the batch size on each node in the distributed training system. Then, we obtain

$$M_{\text{node}} \geqslant \frac{\sum_{\theta\in\mathbb{R}^d} f_t(\theta_t)}{C} \tag{13}$$

Formula (13) means that the training process is convergent when M satisfies Formula (13).

As for the MF in the first-order stochastic gradient algorithms, we could obtain

$$\frac{1-\text{MF}^t}{1-\text{MF}} < \frac{T}{\xi} \tag{14}$$

where T represents the number of iterations. The mathematical derivation is shown in Appendix B.

As Refs. [51, 52] point out, a constant MF in the second-order stochastic gradient algorithms will result in divergence. Therefore, we establish our convergent conditions for distributed training.

(1) The batch size M_{node} on each node in the distributed training satisfies $\frac{\sum_{\theta\in\mathbb{R}^d} f_t(\theta_t)}{C} \leqslant M_{\text{node}} \leqslant M$, where M denotes the size of the training set;

(2) (a) For the first-order stochastic gradient algorithms, $\frac{1-\text{MF}^t}{1-\text{MF}} < \frac{T}{\xi}$ and $\text{MF} < 1$;

(b) For the second-order stochastic gradient algorithms, MF < 1 and MF does not decrease[52].

4.4 Extended experiments on MF

4.4.1 NAG

We also conduct a number of experiments utilizing another first-order optimizer, NAG. In Fig. 7, we show the validation accuracy given a batch size is 1024 and MFs are 0.9, 0.95, 0.975, and 0.99. From Fig. 7, we find that when using a large number of GPUs in distributed training, the validation accuracy drops when the MFs are 0.9, 0.95, and 0.975. When the MF is 0.99, the validation accuracy improves. A large MF helps alleviate the negative influence of a large number of GPUs. This result is consistent with that in Section 4.3.1.

4.4.2 MLP

We train MLP on MNIST of 30 epochs on 8 GPUs to explore the influence of the values of MFs. We use a simple MLP consisting of three fully connected layers, that is, two activation layers and a softmax layer, to identify the output of the MLP. We use the SGD optimizer and set the batch size to 64 and the learning rate to 0.05; the momentum in SGD selected from $\{0.9, 0.95, 0.975, 0.99\}$. Figure 8 shows the test accuracy of the MLP in our experiment. We can identify that the test accuracy increases as the MF increases to a

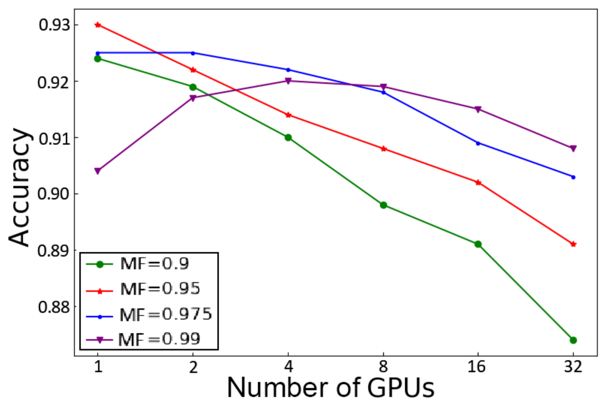


Fig. 7 Validation accuracy of ResNet-50 on CIFAR-10 given different batch sizes on multiple GPUs in the parameter server based on NAG. The hyperparameters are set to be the same as those in the previous SGD and Adam experiments.

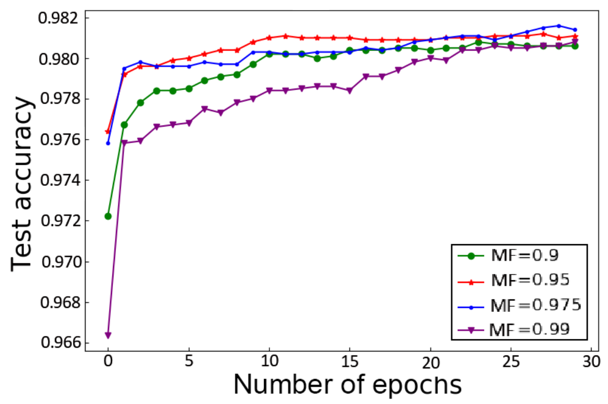


Fig. 8 MLP test results on MNIST of different momentums. The MLP comprises three fully connected layers: two activation layers and a softmax layer.

certain degree. When the MF is 0.975, the test accuracy on MNIST is the highest. However, when the MF is 0.99, the test accuracy drops relative to those given MF values of 0.95 and 0.975. We assume that this phenomenon is caused by a small batch size of 64 and a lack of BN.

4.4.3 RNN

In this section, we present the results of the influence of increasing the MF value in the RNN task. We train LSTM[53] on the PTB dataset. We set the batch size to 64 and the learning rate to 0.01 and then use SGD. The training and validation perplexities are then analyzed. We present our results in Table 4. When MF = 0.975, the smallest training perplexity of 13.40 is noted. When MF = 0.95, the best validation perplexity of 155.31 is observed. From Table 4, we can identify that increasing the MF value within a specific range can reduce the training errors introduced by multiple GPUs.

5 Conclusion

In this work, we discuss different types of stochastic gradient algorithms and distributed algorithms used in our experiments and explore the influence of batch size and the number of GPUs on distributed training performance. We train ResNet-50 on the CIFAR-10 dataset under the parameter server architecture. According to the experimental results, we find that to some degree, we could increase the efficiency of distributed training and reduce the training time by increasing the batch size and increasing the number of GPUs. When the number of GPUs is increased to achieve distributed training, the batch size increases to ensure training convergence. On the other hand, we conclude that for SGD-like optimizers, increasing the MFs alleviates the performance loss caused by the increase in the number of GPUs in distributed training or even improves the accuracy. Increasing MFs not only benefits first-order optimizers, such as SGD, but also suits second-order optimizers, such as Adam. For small tasks, increasing the MF values can relieve the negative effects introduced by multiple GPUs.

Table 4 LSTM's training and validation perplexity on PTB of different MFs.

MF value	Train perplexity	Validation perplexity
0.9	29.93	161.98
0.95	17.04	**155.31**
0.975	**13.40**	158.72
0.99	17.84	173.88

Appendix

A Analysis of Increasing MF$_1$ in Algorithm 2

The detailed analysis of increasing MF$_1$ in Algorithm 2 is as follows. For convenience, we use ϕ to represent $\phi(m_t, v_t, \text{MF}_1, \text{MF}_2)$ in Eq. (A1).

$$\begin{aligned}\xi\cdot\phi = {} & \xi\cdot\frac{m_t\sqrt{1-MF_2}}{(\sqrt{v_t}+\epsilon)(1-MF_1)} = \\ & \xi\cdot\frac{m_tG_1}{G_2(1-MF_1)} = \\ & \frac{G_1}{G_2}\frac{\xi(MF_1m_{t-1}+(1-MF_1)g_t)}{1-MF_1} = \\ & \frac{G_1}{G_2}\left[\frac{\xi(MF_1\cdot m_{t-1})}{1-MF_1}+\xi\cdot g_t\right]\end{aligned} \qquad (A1)$$

$\xi\cdot g_t$ nearly has the same form as that analyzed in Algorithm 1 in Section 3.2. It could also be viewed as constant if we assume $\nabla L(\theta, x)$ constant. Therefore, the whole focus is on $\frac{\text{MF}_1}{1-\text{MF}_1}$. Hence, increasing MF$_1$ has the same effect as increasing MF in the Algorithm 1 in Section 3.2.

B Mathematical Derivation of Eq. (A2)

We show the mathematical derivation of Eq. (A2) here.

Considering v_t in Algorithm 1, we have

$$\begin{aligned}v_t = {} & \text{MF}\cdot v_{t-1}+\xi g_t = \\ & \text{MF}\cdot(\text{MF}\cdot v_{t-2}+\xi g_t)+\xi g_t = \\ & \text{MF}^2\cdot v_{t-2}+\text{MF}\cdot\xi g_t+\xi g_t = \cdots = \\ \text{MF}^t\cdot v_0 + {} & \text{MF}^{t-1}\cdot\xi g_t+\cdots+\text{MF}\cdot\xi g_t+\xi g_t\end{aligned} \qquad (B1)$$

As v_0 is initialized commonly to 0, so we have

$$\begin{aligned}v_t = {} & \text{MF}^{t-1}\cdot\xi g_t+\cdots+\text{MF}\cdot\xi g_t+\xi g_t = \\ & (\text{MF}^{t-1}+\cdots+\text{MF}+1)\cdot\xi g_t = \\ & \frac{1-\text{MF}^t}{1-\text{MF}}\cdot\xi g_t.\end{aligned}$$

According to A4 in Section 4.3.2, we have $g_t \leqslant C$. Therefore,

$$v_t = \frac{1-\mathrm{MF}^t}{1-\mathrm{MF}} \cdot \xi g_t \leqslant \frac{1-\mathrm{MF}^t}{1-\mathrm{MF}} \cdot \xi C.$$

The training process involves T iterations; thus, $v_T \leqslant \frac{1-\mathrm{MF}^T}{1-\mathrm{MF}} \cdot \xi C$.

From the definition of v_t, we have $0 \leqslant v_t < t \cdot C$. This definition is easy to understand. If MF = 1, then, we have the entire gradient in the current iteration. However, such scenario is not realistic in the stochastic gradient method with MF. Therefore, $t \cdot C$ is its upper bound. Then, we get

$$v_T \leqslant \frac{1-\mathrm{MF}^T}{1-\mathrm{MF}} \cdot \xi C \leqslant t \cdot C < T \cdot C$$

Finally, we get

$$\frac{1-\mathrm{MF}^T}{1-\mathrm{MF}} < \frac{T}{\xi},$$

as shown in Section 4.3.2.

Acknowledgment

This work was partially supported by the Major State Research Development Program (No. 2016YFB0201305), the National Key R&D Program of China (No. 2018YFB2101100), the National Natural Science Foundation of China (Nos. 61806216, 61702533, 61932001, and 61872376).

References

[1] Y. Tang, L. J. Yin, Z. N. Zhang, and D. S. Li, Rise the momentum: A method for reducing the training error on multiple GPUs, in *Algorithms and Architectures for Parallel Processing*, S. Wen, A. Zomaya, and L. T. Yang, eds. Cham, Germany: Springer, 2020.

[2] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in *Proc. 2017 IEEE Conf. on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 1800–1807.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, Densely connected convolutional networks, in *Proc. 2017 IEEE Conf. on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 2261–2269.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, SSD: Single shot multibox detector, in *Proc. 14th European Conf. on Computer Vision*, Amsterdam, the Netherlands, 2016, pp. 21–37.

[5] J. F. Dai, Y. Li, K. M. He, and J. Sun, R-FCN: Object detection via region-based fully convolutional networks, in *Proc. 30th Conf. on Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 379–387.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation. in *Proc. 2014 IEEE Conf. on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 580–587.

[7] J. Long, E. Shelhamer, and T. Darrell, Fully convolutional networks for semantic segmentation, in *Proc. 2015 IEEE Conf. on Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 3431–3440.

[8] J. F. Dai, K. M. He, and J. Sun, Instance-aware semantic segmentation via multi-task network cascades, in *Proc. 2016 IEEE Conf. on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 3150–3158.

[9] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in *Proc. 31st AAAI Conf. on Artificial Intelligence*, San Francisco, CA, USA, 2017.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. A. Ma, Z. H. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, *Int. J. Comput. Vis.*, vol. 115, no, 3, pp. 211–252, 2015.

[11] Z. Qin, Z. N. Zhang, X. T. Chen, C. J. Wang, and Y. X. Peng, Fd-mobilenet: Improved mobilenet with a fast downsampling strategy, in *Proc. 2018 25th IEEE Int. Conf. on Image Processing*, Athens, Greece, 2018, pp. 1363–1367.

[12] D. S. Li, Z. Q. Lai, K. S. Ge, Y. M. Zhang, Z. N. Zhang, Q. L. Wang, and H. M. Wang, HPDL: Towards a general framework for high-performance distributed deep Learning, in *Proc. 2019 IEEE 39th Int. Conf. on Distributed Computing Systems*, Dallas, TX, USA, 2019.

[13] F. Tong and X. L. Liu, Samples selection for artificial neural network training in preliminary structural design, *Tsinghua Science and Technology*, vol. 10, no. 2, pp. 233–239, 2005.

[14] Z. Y. Hu, D. S. Li, and D. K. Guo, Balance resource allocation for spark jobs based on prediction of the optimal resource, *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 487–497, 2020.

[15] L. Guan, T. Sun, L. B. Qiao, Z. H. Yang, D. S. Li, K. S. Ge, and X. C. Lu, An efficient parallel and distributed solution to nonconvex penalized linear SVMs, *Front. Inf. Technol. Electron. Eng.*, vol. 21, no. 4, pp. 587–603, 2020.

[16] K. S. Ge, H. Y. Su, D. S. Li, and X. C. Lu, Efficient parallel implementation of a density peaks clustering algorithm on graphics processing unit, *Front. Inf. Technol Electron. Eng.*, vol. 18, no. 7, pp. 915–927, 2017.

[17] M. Li, D. G. Andersen, J. Woo Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Y. Su, Scaling distributed machine learning with the parameter server, in *Proc. 11th USENIX Symposium on Operating Systems Design and Implementation*, Broomfield, CO, USA, 2014.

[18] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Q. Jia, and K. M. He, Accurate, large minibatch SGD: Training imagenet in 1 hour, arXiv preprint arXiv: 1706.02677, 2017.

[19] L. Shen, P. Sun, Y. T. Wang, W. Liu, and T. Zhang, An algorithmic framework of variable metric over-relaxed hybrid proximal extra-gradient method, in *Proc. 35th Int. Conf. on Machine Learning*, Stockholm, Sweden, 2018.

[20] L. Shen, W. Liu, G. Z. Yuan, and S. Q. Ma, GSOS: Gauss-Seidel operator splitting algorithm for multi-term nonsmooth convex composite optimization, in *Proc. 34th Int. Conf. on Machine Learning*, Sydney, Australia, 2017, pp. 3125–3134.

[21] X. Wang, S. Q. Ma, D. Goldfarb, and W. Liu, Stochastic quasi-Newton methods for nonconvex stochastic optimization, *SIAM J. Optim.*, vol. 27, no. 2, pp. 927–956, 2017.

[22] Y. You, Z. Zhang, C. J. Hsieh, J. Demmel, and K. Keutzer, ImageNet training in minutes, in *Proc. 47th Int. Conf. on Parallel Processing*, Eugene, OR, USA, 2018, pp. 1–10.

[23] Y. You, I. Gitman, and B. Ginsburg, Large batch training of convolutional networks, arXiv preprint arXiv: 1708.03888, 2017.

[24] T. Akiba, S. Suzuki, and K. Fukuda, Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes, arXiv preprint arXiv: 1711.04325, 2017.

[25] L. Balles, J. Romero, and P. Hennig, Coupling adaptive batch sizes with learning rates, arXiv preprint arXiv: 1612.05086, 2016.

[26] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, On large-batch training for deep learning: generalization gap and sharp minima, in *Proc. 5th Int. Conf. on Learning Representations*, Toulon, France, 2017.

[27] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Deep residual learning for image recognition. in *Proc. 2016 IEEE Conf. on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[28] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features from Tiny Images*, Toronto, Canada: University of Toronto, 2009.

[29] S. Ghadimi, G. Lan, and H. C. Zhang, Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization, *Math. Program.*, vol. 155, nos. 1&2, pp. 267–305, 2016.

[30] S. L. Smith and Q. V. Le. A Bayesian perspective on generalization and stochastic gradient descent. in *Proc. 6th Int. Conf. on Learning Representations*, Vancouver, Canada, 2017.

[31] D. Masters and C. Luschi, Revisiting small batch training for deep neural networks, arXiv preprint arXiv: 1804.07612, 2018.

[32] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, arXiv preprint arXiv: 1404.5997, 2014.

[33] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina, Entropy-SGD: Biasing gradient descent into wide valleys, in *Proc. 5th Int. Conf. on Learning Representations*, Toulon, France, 2016.

[34] Q. X. Li, C. Tai, and W. E. Stochastic modified equations and adaptive stochastic gradient algorithms, in *Proc. 34th Int. Conf. on Machine Learning*, Sydney, Australia, 2017, pp. 2101–2110.

[35] F. Y. Zou, L. Shen, Z. Q. Jie, J. Sun, and W. Liu, Weighted adagrad with unified momentum, arXiv preprint arXiv: 1808.03408, 2018.

[36] N. Qian, On the momentum term in gradient descent learning algorithms, *Neural Netw.*, vol. 12, no, 1, pp. 145–151, 1999.

[37] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization. in *Proc. 3rd Int. Conf. on Learning Representations*, San Diego, CA, USA, 2015.

[38] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[39] T. Tieleman and G. Hinton, Divide the gradient by a running average of its recent magnitude, *COURSERA*: *Neural Netw. Mach. Learn.*, vol. 4, pp. 26–30, 2012.

[40] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$, *Soviet Math. Dokl.*, vol. 27, no. 2, pp. 372–376, 1983.

[41] J. M. Chen, X. H. Pan, R. Monga, S. Bengio, and R. Jozefowicz, Revisiting distributed synchronous SGD, arXiv preprint arXiv: 1604.00981, 2016.

[42] L. Bottou, F. E. Curtis, and J. Nocedal, Optimization methods for large-scale machine learning, https://doi.org/10.1137/16M1080173, 2018.

[43] S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey, Three factors influencing minima in SGD, arXiv preprint arXiv: 1711.04623, 2017.

[44] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proc. 32nd Int. Conf. on Machine Learning*, Lille, France, 2015, pp. 448–456.

[45] S. L. Smith, P. J. Kindermans, C. Ying, and Q. V. Le, Don't decay the learning rate, increase the batch size, in *Proc. 6th Int. Conf. on Learning Representations*, Vancouver, Canada, 2017.

[46] T. Q. Chen, M. Li, Y. T. Li, M. Lin, N. Y. Wang, M. J. Wang, T. J. Xiao, B. Xu, C. Y. Zhang, and Z. Zhang, MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems, arXiv preprint arXiv:1512.01274, 2015.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE*, vol. 86, no, 11, pp. 2278–2324, 1998.

[48] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, Building a large annotated corpus of English: The Penn Treebank, *Comput. Linguist.*, vol. 19, no, 2, pp. 313–330, 1993.

[49] I. Sutskever, J. Martens, G. Dahl and G. Hinton, On the importance of initialization and momentum in deep learning, in *Proc. 30th Int. Conf. on Machine Learning*, 2013, pp. 1139–1147.

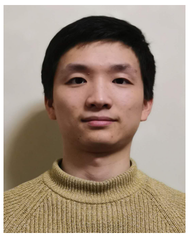
[50] S. Ghadimi and G. H. Lan, Stochastic first- and zeroth-order methods for nonconvex stochastic programming, *SIAM J. Optim.*, vol. 23, no. 4, pp. 2341–2368, 2013.

[51] F. Y. Zou, L. Shen, Z. Q. Jie, W. Z. Zhang, and W. Li, A sufficient condition for convergences of Adam and RMSProp, in *Proc. 2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, Long Beach, CA, USA, 2019.

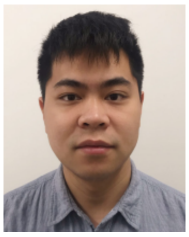
[52] S. J. Reddi, S. Kale, and S. Kumar, On the convergence of Adam and beyond, in *Proc. 6th Int. Conf. on Learning Representations*, Vancouver, Canada, 2018.

[53] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

Yu Tang received the BS degree from National University of Defense Technology, China in 2018, where he is currently pursuing the master degree. His current research interests include distributed machine learning and Alternating Direction Method of Multipliers (ADMM).



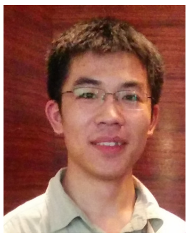
Zhigang Kan received the BEng degree from National University of Defense Technology, China in 2017. His research interests include event extraction and natural language processing.



Lujia Yin received the MS and BS degrees in computer science from National University of Defense Technology (NUDT) in 2017 and 2014, respectively. He is currently a PhD student at the National Key Laboratory for Parallel and Distributed Processing of NUDT. His current research interests include deep learning in cloud environment and Intel CPU deep learning accelerating.

Zhiquan Lai received the PhD, MS, and BS degrees in computer science from National University of Defense Technology in 2015, 2010, and 2008, respectively. He is currently an assistant researcher at the National Key Laboratory for Parallel and Distributed Processing of NUDT. He worked as a research assistant at Department of Computer Science, the University of Hong Kong from Oct. 2012 to Oct. 2013. His current research interests include high-performance system software, distributed machine learning, and power-aware computing.

Zhaoning Zhang received the PhD and MS degrees from National University of Defense Technology, China in 2014 and 2009, respectively. He is currently an associate researcher with the College of Computer, NUDT. His research interests primarily focus on computer vision, deep learning acceleration, and distributed computing.



Linbo Qiao received the PhD, MS, and BS degrees in computer science and technology from National University of Defense Technology, China in 2017, 2012, and 2010, respectively. Now, he is an assistant researcher at the National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, China. He worked as a research assistant at Chinese University of Hong Kong from May 2014 to Oct. 2014. His research interests include structured sparse learning, online and distributed optimization, and deep learning for graph and graphical models.

Dongsheng Li received the PhD degree in computer science and technology from National University of Defense Technology in 2005. He is a professor and doctoral supervisor at the College of Computer, National University of Defense Technology. He was awarded the Chinese National Excellent Doctoral Dissertation in 2008. His research interests include distributed systems, cloud computing, and big data processing.