Received September 23, 2019, accepted October 21, 2019, date of publication October 25, 2019, date of current version November 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2949611

Trend-Smooth: Accelerate Asynchronous SGD by Smoothing Parameters Using Parameter Trends

GUOXIN CUI[1,2], JIAFENG GUO[1], (Member, IEEE), YIXING FAN[1], YANYAN LAN[1], (Member, IEEE), AND XUEQI CHENG[1], (Member, IEEE)

[1]CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China

Corresponding authors: Guoxin Cui (cuiguoxin@ict.ac.cn) and Jiafeng Guo (guojiafeng@ict.ac.cn)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61425016, Grant 61722211, Grant 61773362, and Grant 61872338, in part by the Youth Innovation Promotion Association CAS under Grant 20144310 and Grant 2016102, in part by the National Key Research and Development Program of China under Grant 2016QY02D0405, and in part by the Foundation and Frontier Research Key Program of Chongqing Science and Technology Commission under Grant cstc2017jcyjBX0059.

ABSTRACT Stochastic gradient descent(SGD) is the fundamental sequential method in training large scale machine learning models. To accelerate the training process, researchers proposed to use the asynchronous stochastic gradient descent (A-SGD) method in model learning. However, due to the stale information when updating parameters, A-SGD converges more slowly than SGD in the same iteration number. Moreover, A-SGD often converges to a high loss value and results in lower model accuracy. In this paper, we propose a novel algorithm called Trend-Smooth which can be adapted to the asynchronous parallel environment to overcome the above problems. Specifically, Trend-Smooth makes use of the parameter trend during the training process to shrink the learning rate of some dimensions where the gradients' directions are opposite to the trends of parameters. Experiments on MNIST and CIFAR-10 datasets confirm that Trend-Smooth can accelerate the convergence speed in asynchronous training process. The test accuracy that Trend-Smooth achieves is shown to be higher than other asynchronous parallel baseline methods, and is very close to the SGD method. Moreover, Trend-Smooth can also be combined with other adaptive learning rate methods(like Momentum, RMSProp and Adam) in the asynchronous parallel environment to promote their performance.

INDEX TERMS Parameter trend, asynchronous SGD, accelerate training.

I. INTRODUCTION

Stochastic gradient descent(SGD) is the most widely used and fundamental sequential method in training machine learning models recently. In each iteration, it uses a small subset of the whole dataset to compute gradients and use them to update model parameters. Usually, SGD can be implemented using a single machine and it needs to take a large number of iterations to converge during the learning process. Despite the slow parameter update frequency due to limited computing power, SGD converges well. To accelerate the convergence, many gradient-based methods have been proposed, such as Momentum [1], RMSProp [2], and Adam [3] methods. Although these methods do help accelerate the training, the convergence still needs a lot of time due to the huge computation amount of gradient and the large iteration number.

The associate editor coordinating the review of this manuscript and approving it for publication was Stéphane Zuckerman.

As the computational ability of a single machine is always limited, researchers propose asynchronous stochastic gradient descent (A-SGD) method which can leverage the cluster's computing power to accelerate the training. The A-SGD method can be easily implemented using the Parameter Server(PS) [4]–[6] system. In PS system, the whole cluster is composed of the server and workers. The server holds a global copy of the whole parameters and the workers compute gradient of the machine learning model using local training data. In the A-SGD method, the server updates the parameters as long as the gradient from one worker comes. After finishing updating global parameters, the server sends the newest parameters to the worker. The parameter update frequency of the A-SGD method is higher compared to the SGD method. However, the local parameter producing the gradient is often not consistent with the global parameter to be updated, because the server may have updated the global parameter based on other worker's local gradient. This is also known as the stale gradient problem [7]. Due to this reason, the A-SGD method can not perform as good as the SGD method. Often, it converges slowly to a higher loss value and results in lower test accuracy compared to the SGD method.

To further understand the impact of the stale gradient, we have conducted preliminary experiments on both the A-SGD method and the SGD method. We find that the parameter curves of the A-SGD method also have some trends, which is similar to that of the SGD method found in previous work [8]. However, the parameter curves of the A-SGD method become more fluctuated than the curves of the SGD method. Also, the parameter values in A-SGD method change less than the SGD method. These features are both considered to be the consequence of the stale gradient.

Based on these observations, in this paper we propose a novel algorithm called "Trend-Smooth" which can speed up the convergence speed and promote the test accuracy in the asynchronous parallel environment. Specifically, Trend-Smooth computes each parameter's trend using the history of parameters stored on the server. These trends give us the reliability for the current gradient of each dimension. For each dimension which violates the trends, we reduce its effect by shrinking the learning rate of the dimension while keeping the learning rate of other dimensions unchanged. This can cut down the impact of the stale gradient during the training. To verify the effectiveness of the method, we conduct experiments on CIFAR-10 dataset and MNIST dataset based on two existing neural network architectures. On both datasets, Trend-Smooth 's loss decreases faster than other asynchronous parallel methods which means that it can accelerate the training. Also, Trend-Smooth achieves higher test accuracy (very close to the SGD method) compared to other asynchronous parallel baseline methods. Moreover, Trend-Smooth can be combined with other adaptive learning rate methods like these mentioned above(e.g. Momentum, RMSProp and Adam) to promote their performances in asynchronous parallel environment.

The major contributions of this paper include:

- We explore the impact of the stale gradient on the A-SGD method by observing the parameter curves in comparison to the SGD method.
- We propose a novel asynchronous parallel method named "Trend-Smooth" which makes use of the parameter trend to shrink the learning rate of the gradient dimensions in which the gradient values violate the parameter trend.
- We conduct experiments to verify that Trend-Smooth can speed up the training convergence, and achieve higher test accuracy (very close to the SGD method) compared to other asynchronous parallel methods.

The rest of the paper are organized as follows: After introducing the related work in Section II, we explore the impact of the stale gradient on the A-SGD method in comparison to the SGD method in Section III. Based on these observations, we propose the Trend-Smooth method in Section IV. Section V presents the experimental results to verify the effectiveness of the proposed Trend-Smooth method and the conclusions are drawn in Section VI.

II. RELATED WORK

A. SGD AND ITS VARIANTS

In this section, we give a detailed introduction to the SGD method and its variants.

The goal of many machine learning (including the deep learning) algorithms can be expressed via an "loss function", which captures the properties of the learned model, such as the error in terms of the training data and the complexity of the learned model. Given a set of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ and y_i are the feature vector and the label for the i-th instance, respectively, and N is the number of training instances, the machine learning algorithm typically minimizes the loss function of the model:

$$\min L(\mathbf{w}) = \min \sum_{i=1}^{N} \ell(\mathbf{x}_i, y_i; \mathbf{w}) + \Omega(\mathbf{w}),$$

where ℓ represents the prediction error on the training data and regularizer Ω penalizes the model complexity. In general, there is no closed-form solution for L. Instead, learning starts from an initial model. It iteratively refines the model by executing the following formular:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta * \mathbf{u},$$

where t means the current iteration number and $\mathbf{u}$ means the update vector. The training process stops when a (near) optimal solution of L is found or the model is considered to be converged.

The gradient descent (GD) method, where $\mathbf{u}$ is equal to $\nabla L(\mathbf{w})$, can get ϵ error by using $\mathcal{O}(log\frac{1}{\epsilon})$ steps [9], [10] under certain circumstances. Second-order optimization method like Newton method can achieve ϵ error by using $\mathcal{O}(loglog\frac{1}{\epsilon})$ steps [9] under certain circumstances. There are also works like stochastic quasi-Newton methods [11]–[14] to solve the optimization problem.

When the dataset is very large, getting the gradient of the whole training data (like the GD method) is not computationally affordable. To overcome this issue, stochastic gradient descent(SGD) method uses a single training sample $\mathbf{x}_i$ and $\mathbf{y}_i$, which is randomly chosen from the whole dataset, to finish one iteration:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta * \nabla L(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w})$$

To make the gradient more accurate and leverage current computing hardware like GPU efficiently, mini-batch SGD is the most widely used algorithm. It randomly selects a mini-batch from the whole dataset and uses the gradient from the mini-batch to update the parameter:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta * \frac{1}{M} \sum_{i=1}^{M} \nabla L(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w}),$$

where M (M $\ll$ N) is the batch size. The convergence of the SGD method has been studied extensively in the stochastic approximation literature. It has been proved that when using diminishing learning rate and when the optimization problem is convex, SGD can get ϵ error by taking $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ training steps [15]. When the loss function is differentiable with Lipschitz gradient, SGD can take $\mathcal{O}(\frac{1}{\epsilon})$ steps [9], [10] to reach error ϵ by choosing suitable fixed learning rate. In this paper, we use mini-batch SGD to conduct our experiments.

The SGD method uses a constant learning rate for all dimensions of parameters. How to choose the learning rate is an important problem as the learning rate often has a great impact on the convergence of the learning progress. To solve these problems, researchers have proposed a large number of adaptive learning rate methods based on SGD [1]–[3], [16]. In the following, we will introduce several important adaptive learning rate methods.

1) The Momentum SGD method [1] is a method that accelerates SGD in relevant direction by adding a fraction γ of the updated vector of past time step to the current update vector. The update formulas show as follows:

$$\begin{aligned}\mathbf{u}_t &= \gamma * \mathbf{u}_{t-1} + \eta * \mathbf{g} \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \mathbf{u}_t\end{aligned}$$

2) Adagrad method [16] adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data. Its update rules are as follows:

$$\begin{aligned}\mathbf{G}_t &= \mathbf{G}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}} \odot \mathbf{g}_t,\end{aligned}$$

where $\odot$ denotes element-wise product. Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This, as a result, makes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge [17].

3) RMSProp method [2] replaces $\mathbf{G}_t$ in Adagrad by an exponentially decaying average of squared gradients to avoid the final learning rate being infinitesimally small. Its update rules are as follows:

$$\begin{aligned}\mathbf{G}_t &= \beta * \mathbf{G}_{t-1} + (1-\beta) * \mathbf{g}_t \odot \mathbf{g}_t \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}} \odot \mathbf{g}_t\end{aligned}$$

4) Adaptive momentum estimation (Adam) method [3] is now the most widely used optimization method in training machine learning model [18]. It keeps an exponentially decaying average of past gradients $\mathbf{m}_t$(the first moment estimate) and an exponentially decaying average of past squared gradients $\mathbf{v}_t$(the second moment estimate). As $\mathbf{m}_t$ and $\mathbf{v}_t$ are initialized as zero, they are biased towards zero. To counteract these biases, Adam uses bias-corrected first and second moment estimates. The whole update rules are as follows:

$$\begin{aligned}\mathbf{m}_t &= \beta_1 * \mathbf{m}_{t-1} + (1-\beta_1) * \mathbf{g}_t \\ \mathbf{v}_t &= \beta_2 * \mathbf{v}_{t-1} + (1-\beta_2) * \mathbf{g}_t^2 \\ \overline{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t} \\ \overline{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_2^t} \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\eta}{\sqrt{\overline{\mathbf{v}}_t} + \epsilon} * \overline{\mathbf{m}}_t\end{aligned}$$

The above adaptive learning rate methods ((1) to (4)) focus on making the training process converge faster compared to the SGD method. The adaptive learning rate methods perform well on the training dataset, however, they do not generalize well on the test dataset [18]. Works in natural language processing [19] and computer vision [20] also reflected this fact. In this paper, we use the Momentum method((1)), the RMSProp method((3)), the Adam method((4)) in asynchronous parallel environment as baselines to show that the Trend-Smooth combined versions can promote their performances.

In asynchronous paralle environment, reference [7] proposed Delay Compensated ASGD(DC-ASGD) to compensate the delay of the gradient. It made the optimization behaviour of A-SGD closer to that of the sequential SGD by leveraging the Taylor expansion of the gradient function and efficient approximation to the Hessian matrix of the loss function. The global parameter update rules are shown as follows:

$$\mathbf{w}_{t+\tau+1} = \mathbf{w}_{t+\tau} - \eta * (\mathbf{g}_t + \lambda * \mathbf{g}_t \odot \mathbf{g}_t \odot (\mathbf{w}_{t+\tau} - \mathbf{w}_t)),$$

where τ denotes the time delay between the gradient $\mathbf{g}_t$ and the parameter $\mathbf{w}_{t+\tau}$. In this paper, we use DC-ASGD as one of our baseline methods.

B. PARAMETER SERVER

In literature, several systems have been used to train the machine learning model. For example, massage passing interface (MPI) [21], [22] can be used to construct high performance system. Despite the high performance it achieves, its scalability and fault tolerance ability is poor. There exist many general-purpose distributed systems for machine learning applications. Mahout [23] based on Hadoop [24] and MLI [25] based on Spark [26] both adopt the iterative MapReduce [27] framework. These systems have no global state and have high fault tolerance ability. They can scale well to a few hundreds of nodes, primarily on dedicated research clusters. However, they only support synchronous parallel scheme, which has the strongest model consistency.

To make the model consistency more flexible, researchers proposed Parameter Server(PS) architecture [4]–[6]. PS divides machines of the cluster into two categories, namely server(s) and workers. The server holds a global copy of the parameters. It receives gradients from all workers to update the parameters. Then, it sends the updated parameters

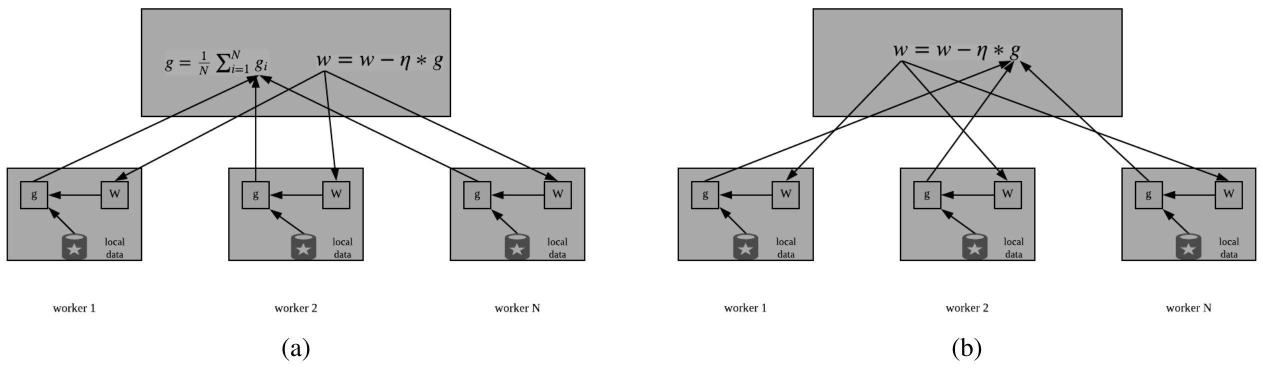


FIGURE 1. **(a) Workflow of the BSP mode. (b) Workflow of the ASP mode.**

back to each corresponding worker. Each worker computes the gradients based on local data and sends the gradients to the server. The PS will loop the two stages continuously.

The two commonly used parallel modes are bulk synchronous parallel(BSP) [28], [29] and asynchronous parallel(ASP) [6], [30]–[32]. Each mode has its own advantages and disadvantages. Firstly, the BSP mode has no stale gradient information, so it converges stably and faster compared to the ASP mode in the same number of iterations. However, all workers have to wait for each other to begin the next iteration. The BSP's updating frequency is slow as it averages updates from all workers as a single update. The SGD method can be seen as a special case of the BSP mode when the PS system has only one worker. Adding more workers in PS just makes the batch size larger compared to the SGD method. In this paper, we use the single-machine SGD method to conduct our experiments.

In ASP mode, the server updates the parameters as long as it receives gradients from a worker. So the ASP mode has faster parameter updating frequency than BSP mode. However, due to stale information(delayed gradient) [33]–[35], it often converges more slowly than the BSP mode (SGD) in the same number of iterations and the loss value remains larger in the later training process which may make the test accuracy lower [35]–[39]. The workflows of the BSP mode and the ASP mode are shown in Fig. 1. Recently, machine learning tools such as TensorFlow [9] can support BSP and ASP implementations easily.

Besides the PS architecture in which the server and the workers are connected to transport information, there were works [40]–[43] that used ring-based all reduce method to leverage the cluster's computing power. By arranging the machines in a logical ring, ring-based all-reduce method solved the problem that the server may become the bottleneck of the system because of the large volumes of data to transport.

C. PARAMETER TRENDS

In reference [8], it is found that when training neural networks, the evolution of the parameters had some trend during the training process. The important findings were that a small subset of parameters undergo massive changes compared to the rest, and these parameters were observed to follow a predictable trend: they either have an obviously increasing trend or an obviously decreasing trend during the training process. With this notice, reference [8] trained a neural network I using training data(parameter history during the training process) collected from training another simple network. Then they used the network I to predict parameter values at specific iterations. Experiment results showed that this method could accelerate the training process and promote test accuracy.

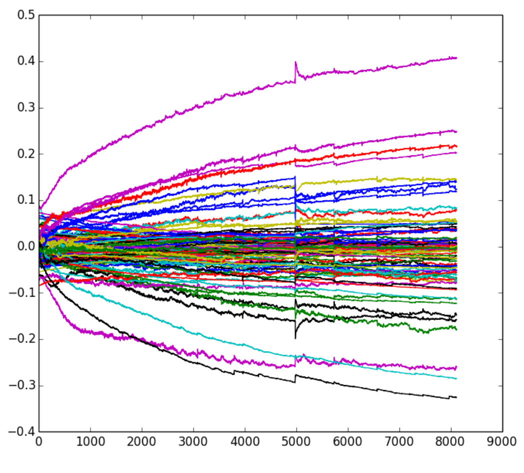
III. PRELIMINARY EXPERIMENTS

To explore the impact of the stale gradient on the A-SGD method in comparison to the SGD method, we conduct experiments on CIFAR-10 dataset with a 5-layer convolutional neural network. The neural network's first two layers are convolution layers and the other three layers are fully-connected layers. We randomly choose 100 dimensions of parameters in the first convolutional layer and log the value of parameters of these dimensions at every step during the training.

For both the SGD and the A-SGD methods, we use a learning rate of 0.1 and the batch size is set to 64. We use 4 workers in the A-SGD method. Fig. 2 shows the parameter curves of the SGD method and the A-SGD method.

From the figure we can see that the parameter curves in the A-SGD method also have some trends, similar to the SGD method found in reference [8]. However, the parameter curves in the A-SGD method are more fluctuated than those of the SGD method. To elaborate this more clearly, we split each line every 1000 consecutive iterations to get small episodes. For each episode, we use least-squares regression to fit it and compute the standard error which is the root of the square error. Also we compute the value changes of these parameters. The value change for one parameter is the absolute difference between its final value and its initial value. Table 1 shows the sums of the standard errors and value changes of all curves in both methods.

From Table 1 we see that the A-SGD method has a higher standard error and fluctuates more than the SGD method.



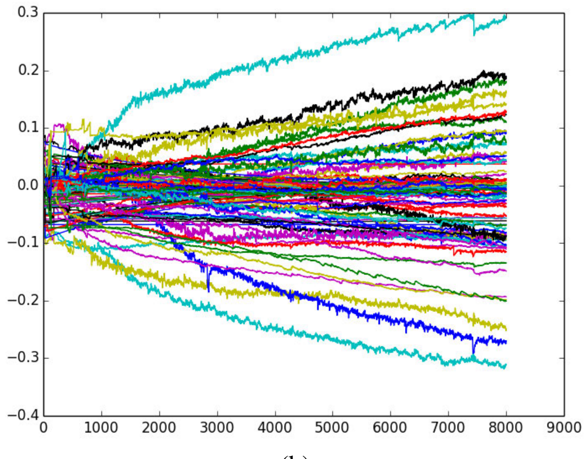


FIGURE 2. **(a) 100 randomly chosen parameter trend trained by the SGD method. (b) 100 randomly chosen parameter trends trained by the A-SGD method with 4 workers.**

TABLE 1. **The summed standard error and values changed of the SGD and the A-SGD methods.**

Methods	Std Err	Values Changed
SGD	1.1e-4	6.26
A-SGD	5.1e-4	3.31

In this process, lots of time in A-SGD method is wasted on the fluctuation and as a result, the A-SGD method's parameters change less than the SGD method.

The above phenomenon clearly reflects the impact of the stale gradient in the A-SGD method and this may be the reason why the A-SGD method converges worse than the SGD method.

IV. OUR APPROACH: TREND-SMOOTH

In this section, we will introduce the Trend-Smooth algorithm. As discussed in previous section, the parameter curves differ significantly between the SGD method and the A-SGD method, in that the latter are more fluctuated than the former. Due to the fluctuation, the A-SGD method's parameters change less than the SGD method. Based on these observations, we try to leverage the parameter trend to accelerate the training process. The whole idea is as follows. Since we have the whole history of the parameters, we can calculate the parameter trend and use the trend information to judge the gradient. We will keep the gradient values that accord with the trend while shrinking the gradient values that violate the trend. By doing this, we can cut down the impact of the fluctuated curves.

Before introducing the Trend-Smooth algorithm, we first specify the symbols we use. The symbols are described in the Table 2.

TABLE 2. **Symbols used in Trend-Smooth.**

Symbols	Explanation
$\mathbf{w}$	Global parameters stored on server
$\mathbf{g}^m$	Local gradients computed by worker m
$\mathbf{g}_i$	The i-th dimension of $\mathbf{g}$
$\mathbf{H}$	Stack that stores the history of global parameters on the server
L	Max size of $\mathbf{H}$
$\mathbf{u}$	Update value of $\mathbf{w}$
trend	Parameter trends
t	Time step in server
α	Value to shrink gradient

A. HOW TO COMPUTE THE TRENDS

As we aim to leverage the parameter trends to update the global parameters, it is of great importance to get the parameter trends in an efficient way. In order to reduce the time cost in calculating the parameter trends, in the Trend-Smooth method, we use the difference between the latest parameter copy and the oldest parameter copy to measure the parameter trend. Specifically, we maintain a stack $\mathbf{H}$ on the parameter server which holds the past history of parameters. $\mathbf{H}$'s max length is set to L. We first get the newest parameter copy in $\mathbf{H}$ as e and the oldest parameter copy in $\mathbf{H}$ as f. Then we obtain the parameter trend by subtracting f from e. The whole process is described in the Algorithm 1.

Algorithm 1 Trend-Smooth:GetTrend

```
Input:H
Output:trend
1: $\mathbf{e} \leftarrow \text{end}(\mathbf{H})$
2: $\mathbf{f} \leftarrow \text{front}(\mathbf{H})$
3: $\textbf{trend} \leftarrow \mathbf{e} - \mathbf{f}$
4: return trend
```

B. HOW TO UPDATE PARAMETERS BASED ON THE TREND

Suppose the server now gets the trend and the gradient $\mathbf{g}$ from a worker, how does it perform an update using this information? In Trend-Smooth, we use a simple and efficient rule to get the update. Specifically, we use an additional hyper-parameter α to penalize the dimensions where the signs of the gradients run counter to the sign of the trend. For those dimensions of the gradients whose signs are consistent with the trend, we keep the learning rate unchanged.

The algorithm is depicted in the Algorithm 2. It is worth noting that in the 4th line of Algorithm 2, we have compared the sign of $trend_i$ and g_i to determine whether to adjust the learning rate. By using the trend to penalize the un-accorded dimensions of gradients, the parameter curves will be more smooth and change rapidly towards the trend directions. The gradient values that do not accord with the trend will produce less effect on the training process. Since we do not change the signs of gradients, the loss function still decreases. At the same time, it would not hinder the parameters from converging or changing their trends during training process.

Algorithm 2 Trend-Smooth:GetUpdate

```
Input:trend, g
Output:u
1: $D \leftarrow$size($\mathbf{g}$)
2: $\mathbf{u} \leftarrow \mathbf{0}$
3: for $i$ from 1 to $D$ do
4:     if sign($\mathbf{trend}_i$) == sign($\mathbf{g}_i$) then
5:         $\mathbf{u}_i \leftarrow \mathbf{g}_i * \alpha$
6:     else
7:         $\mathbf{u}_i \leftarrow \mathbf{g}_i$
8:     end if
9: end for
10: return u
```

C. THE WHOLE LEARNING PROCESS

The whole process of Trend-Smooth is shown in Algorithm 3 and Algorithm 4. Every worker pulls the latest parameters $\mathbf{w}$ on the server before computing gradients $\mathbf{g}$ using local data $\mathbf{D}$. After worker m finishes computing gradient g^m, it sends g^m to the parameter server. When server receives g^m from worker m, it first computes the trends based on the past parameter history $\mathbf{H}$. Then server uses these trends and g^m to get parameter update $\mathbf{u}$ to update $\mathbf{w}$. After the parameters have been updated, the server pushes the latest parameter $\mathbf{w}$ to history stack $\mathbf{H}$ to compute the trends of the next iteration. If $\mathbf{H}$'s size exceeds the max size L, $\mathbf{H}$ will pop out the oldest parameter. Fig. 3 shows the workflow of the whole system.

Algorithm 3 Trend-Smooth:Worker m

```
1: while NOT FINISHED do
2:     Pull $\mathbf{w}_t$ from server
3:     Compute $\mathbf{g}^m \leftarrow \nabla f(\mathbf{w}_t)$ using local data
4:     Send $\mathbf{g}^m$ to server
5: end while
```

V. EXPERIMENT

A. EXPERIMENTAL SETTINGS

To test the performance of the proposed Trend-Smooth algorithm, we conduct experiments on two datasets:

Algorithm 4 Trend-Smooth:Server Side

```
Input:H, $L$, $t$
1: while NOT FINISHED do
2:     if receives $\mathbf{g}^m$ then
3:         trend $\leftarrow$ GetTrend($\mathbf{H}$, $s$)
4:         $\mathbf{u} \leftarrow$ GetUpdate($\mathbf{trend}$, $\mathbf{g}^m$)
5:         $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta * \mathbf{u}$
6:         $t \leftarrow t + 1$
7:         push $\mathbf{w}_t$ to $\mathbf{H}$
8:         if size($\mathbf{H}$) > L then
9:             pop($\mathbf{H}$)
10:        end if
11:    else {receives pull request from worker $m$}
12:        send $\mathbf{w}_t$ to worker $m$
13:    end if
14: end while
```

CIFAR-10 [44] and MNIST [45]. We maintain the max length L of the history stack $\mathbf{H}$ to be 200. We perform grid search on both datasets and the best test performances are obtained by choosing the shrinking parameter α to 0.8.

The baselines we choose include stochastic gradient descent method(SGD), asynchronous parallel stochastic gradient descent method(A-SGD), and asynchronous delay-compensate(A-DC) method in which the hyper-parameter λ is set as 0.04. To eliminate the effect of the different learning rates during training, we also set up another baseline called A-SGD-alpha which is the same to A-SGD except the learning rate is shrunk by α (0.8) for all dimensions of gradients.

To show that Trend-Smooth can be combined with other adaptive learning rate methods, we also do experiments using asynchronous momentum method(A-Momentum), asynchronous RMSProp method(A-RMSProp), asynchronous Adam method (A-Adam) and their Trend-Smooth combined versions on these two datasets.

In the following, we introduce the specific configurations of the MNIST dataset and CIFAR-10 dataset, respectively.

1) MNIST

We conduct our experiments of the proposed Trend-Smooth algorithm on MNIST dataset with an 11-machine cluster which consists of a server and ten workers. Each machine has a 2.3GHz frequency CPU and the machines are connected by a network with 1000Mb/s bandwidth.

The machine learning model we choose to train MNIST is a simple 3-layer fully-connected neural network named $MNIST_3$ in reference [8]. The shapes of parameters of these layers are [784, 256], [256, 256] and [256, 10]. Cross entropy loss is used as the loss function. For all the methods, the batch size is set to 100. We run each method 20000 iterations. To achieve the best performance, each method is fine tuned and the learning rate decays by half every 5000 iterations. The initial learning rate used for A-SGD-alpha is 0.08(0.1*α). We set the learning rates of A-RMSProp and A-Adam as

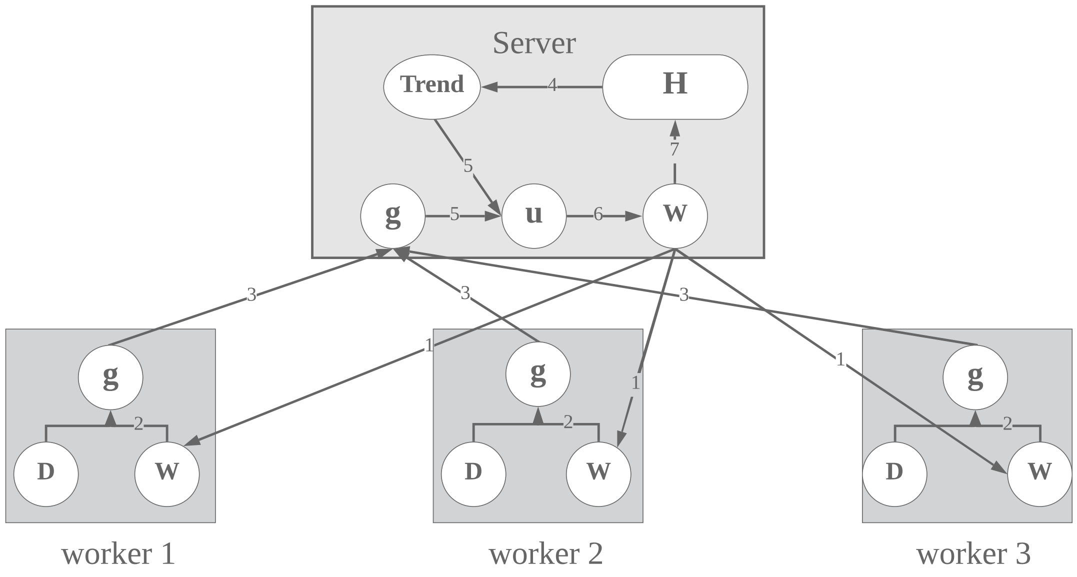


FIGURE 3. Trend-Smooth system architecture. 1. Workers pull the latest parameters w from parameter server. 2. Workers compute gradient g using local data D. 3. Workers send gradient g to the server. 4. Server gets the trend from parameter history. 5. Server uses the trend and current gradient g to compute the update u. 6. Server uses u to update w. 7. Server pushes w to the history stack H for next trend computing.

0.002 in order to run these methods properly. The learning rate of A-Momentum method is set as 0.01 and the momentum efficient is set as 0.9. The initial learning rates for other methods are set as 0.1.

2) CIFAR-10

To test the performances of the proposed Trend-Smooth algorithm on CIFAR-10 dataset, we conduct experiments on a 5 nodes cluster which consists of 1 server and 4 workers. Each node's setting is the same as the one used conducting the MNIST experiment.

The machine learning model tested is a 5-layer neural network named $CIFAR_1$ from reference [8]: the first two layers are convolutional layers with each layers' parameter shape being [5, 5, 3, 64] and [5, 5, 64, 64]. Local response normalization after max-pooling is used [46]. The third and fourth layers are fully connected layers with shapes [2304, 384] and [384, 192], respectively. The last softmax layer is also a fully connected layer with shape [192, 10]. We use the cross entropy as the loss. During the training, the batch size is set as 128 for all methods. We run each method 50000 iterations. To achieve the best performance, we fine tune each method and use a learning rate with a decay of 0.8 every 3000 iterations. The learning rate for A-SGD-alpha method is set as $0.08(0.1{*}\alpha)$. We set the learning rates of A-RMSProp and A-Adam as 0.002 in order to run these methods properly. The learning rate of A-Momentum method is set as 0.01 and the momentum efficient is set as 0.9. The other methods' initial learning rates are set as 0.1.

B. EXPERIMENTAL RESULTS

1) MNIST

The picture (a) in Fig. 4 shows the training loss curves of Trend-Smooth as well as all other baselines. The x-axis indicates the training iterations and y-axis indicates the loss. From the figure, we can see that the loss of the Trend-Smooth decreases faster than those of other asynchronous baseline methods in the beginning of the learning process. Moreover, the Trend-Smooth also achieves much lower loss value compared with all other asynchronous parallel methods, and this value is almost the same with that of the SGD method.

The picture (b) in Fig. 4 shows the loss values with respect to the training time of all methods. The SGD method's loss decreases the slowest among all methods due to the lower sequential parameter update frequency in comparison to other asynchronous parallel methods, while the Trend-Smooth method's loss curve decreases the fastest among all these methods.

The picture (c) in Fig. 4 shows the accuracies of all methods with respect to the training iterations. As we can see, the Trend-Smooth also achieves better performance compared with all the asynchronous baseline methods. Moreover, the result is comparable with the SGD method. We also report the best accuracies in the Table 3. All the asynchronous methods (except the Trend-Smooth method) get very close accuracy, obviously lower than the SGD method. Our Trend-Smooth method, which use the parameter trend to accelerate the learning process in the asynchronous methods, achieves close accuracy to the SGD method.

TABLE 3. Accuracies of all methods on MNIST dataset.

Methods	Accuracies
A-SGD	97.42%
A-SGD-alpha	97.28%
A-DC	97.36%
SGD	98.26%
Trend-Smooth	**98.07%**

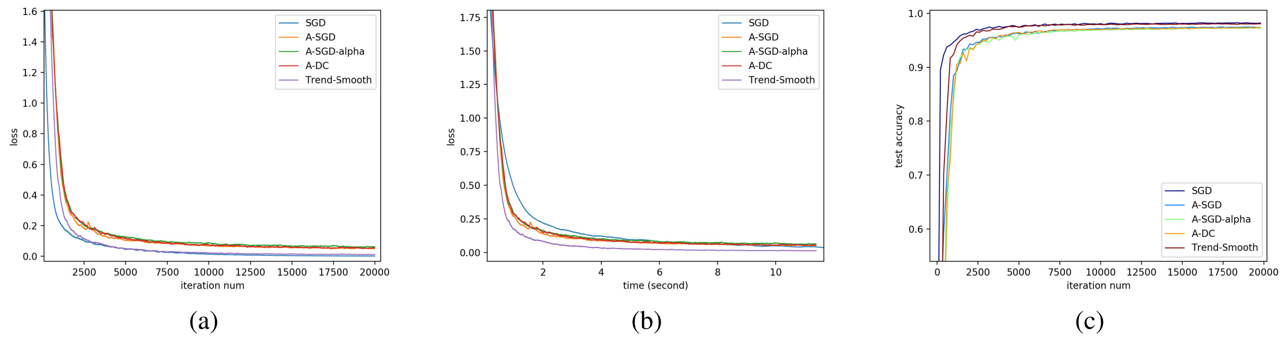


FIGURE 4. (a) Loss value with respect to the iteration number of different methods on MNIST dataset. (b) Loss value with respect to the training time of different methods on MNIST dataset. (c)Test accuracy with respect to the iteration number of different methods on MNIST dataset.

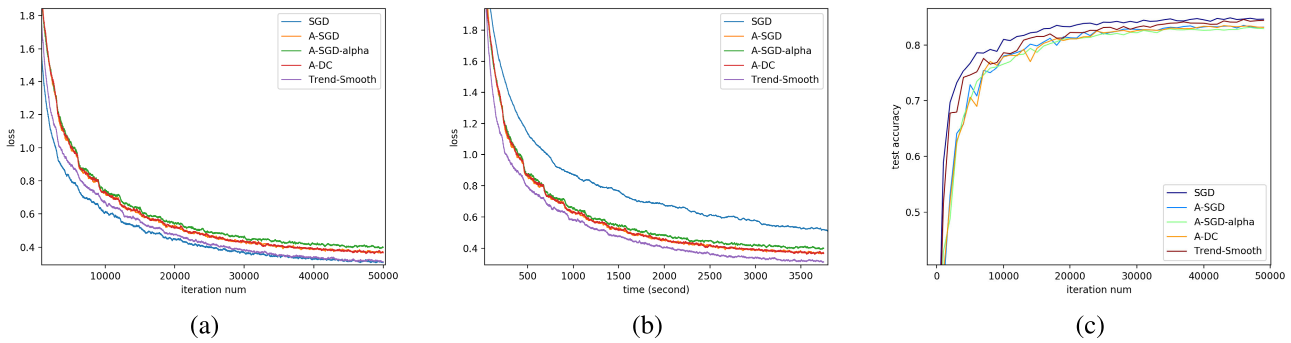


FIGURE 5. (a) Loss value with respect to the iteration number of different methods on CIFAR-10 dataset. (b) Loss value with respect to the training time of different methods on CIFAR-10 dataset. (c) Test accuracy with respect to the iteration number of different methods on CIFAR-10 dataset.

TABLE 4. Accuracies of all methods on CIFAR-10 dataset.

Methods	Accuracies
A-SGD	83.37%
A-SGD-alpha	83.11%
A-DC	83.31%
SGD	84.74%
Trend-Smooth	**84.58%**

TABLE 5. Accuracies on MNIST dataset of adaptive learning rate methods and their Trend-Smooth counterparts.

Methods	Accuracies
A-Momentum	96.53%
A-RMSProp	97.81%
A-Adam	97.32%
Trend-Smooth+ A-Momentum	**97.60%**
Trend-Smooth+ A-RMSProp	**98.30%**
Trend-Smooth+ A-Adam	**97.94%**

2) CIFAR-10

The picture (a) in Fig. 5 shows the loss curves of all methods trained on the CIFAR-10 dataset. From the figure we can see that our Trend-Smooth method converges faster than other asynchronous parallel methods in the early stage of the training process. As the iteration grows, it could achieve nearly the same loss compared with SGD method. This is consistent with the results on the MNIST dataset, which demonstrate the power of the Trend-Smooth method.

The picture (b) in Fig. 5 shows the loss values with respect to the training time of all methods. The SGD method's loss decreases the slowest while the Trend-Smooth method's loss curve decreases the fastest among all these methods, which is consistent with the results on the MNIST dataset. From these results, we can confirm that the Trend-Smooth method can accelerate the training.

The picture (c) in Fig. 5 shows the test accuracies of all the methods with respect to iteration numbers. Table 4 shows the final test accuracies of these methods. From the figure and the table we can draw the same conclusions as is described in the MNIST dataset.

3) TREND-SMOOTH COMBINED WITH ADAPTIVE LEARNING RATE METHODS

The Trend-Smooth algorithm can be combined with other adaptive learning rate methods(e.g. Momentum, RMSProp and Adam) easily. Here we choose the update $\mathbf{u}$ in Trend-Smooth as the gradient $\mathbf{g}$ in the adaptive learning rate methods. Fig. 6 shows the loss curves of the adaptive learning rate methods and their Trend-Smooth counterpart versions on MNIST and CIFAR-10 datasets. We see that the losses of the methods combined with Trend-Smooth decrease faster and get lower values compared to the original methods. Table 5 and Table 6 show that the Trend-Smooth combined

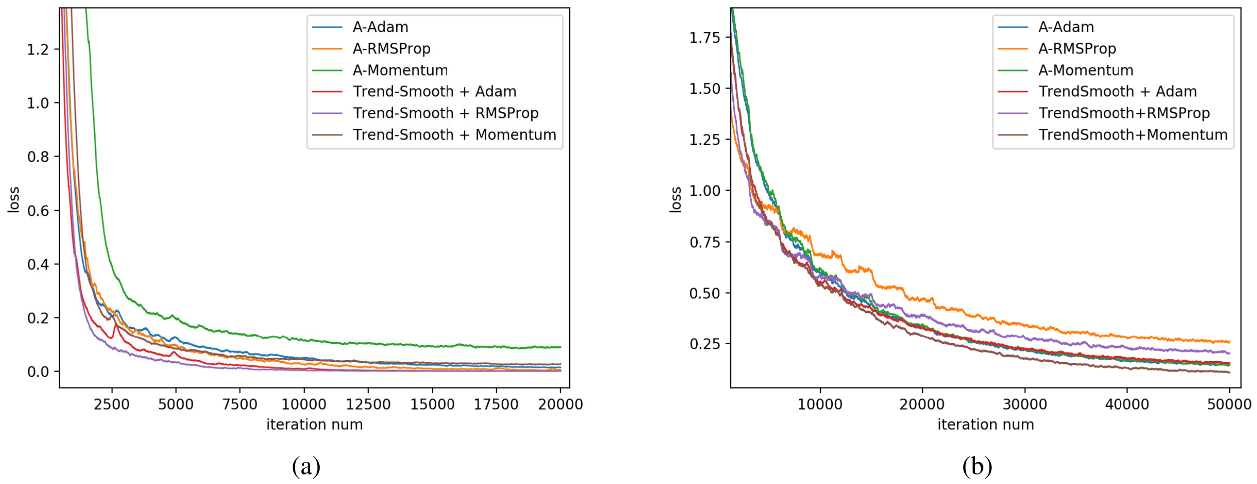


FIGURE 6. **(a) Loss curves of adaptive learning rate methods and their Trend-Smooth combined versions on MNIST dataset. (b) Loss curves of adaptive learning rate methods and their Trend-Smooth combined versions on CIFAR-10 dataset.**

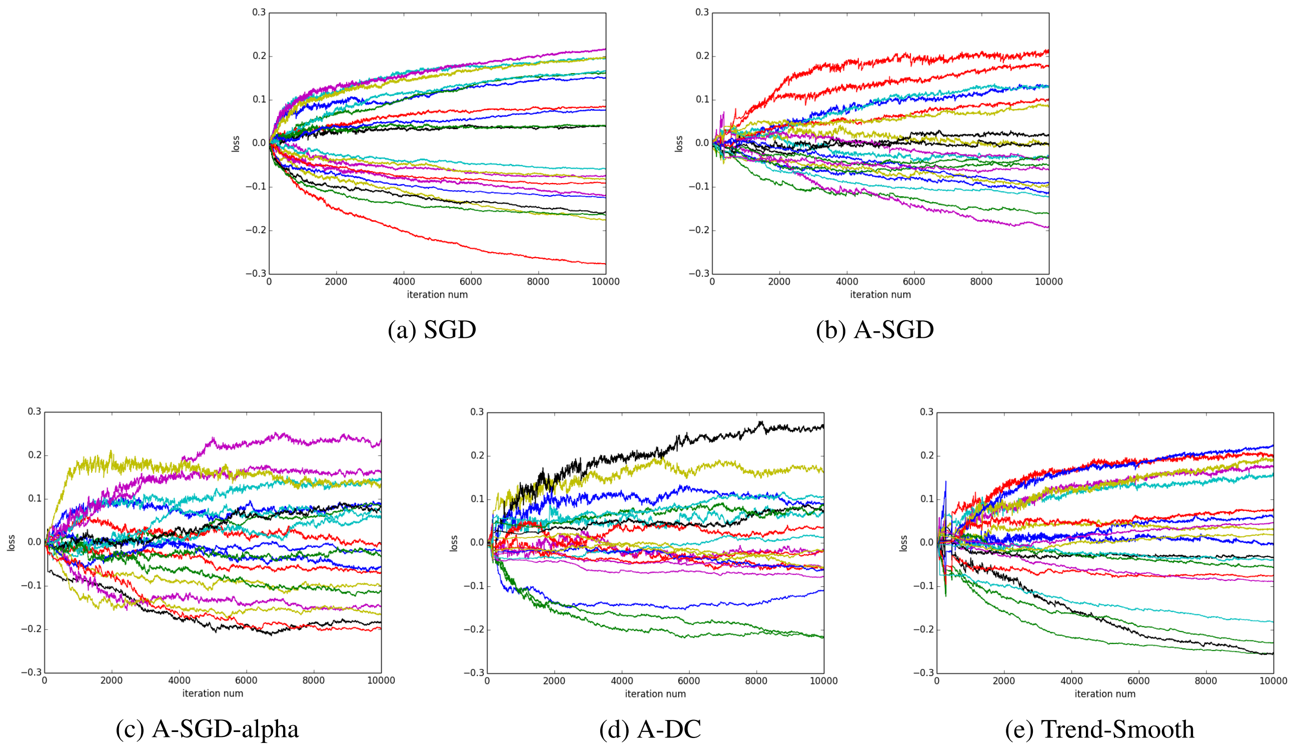


FIGURE 7. **(a) Parameter curves of SGD training method. (b) Parameter curves of A-SGD training method. (c) Parameter curves of A-SGD-alpha training method. (d) Parameter curves of A-DC training method. (e) Parameter curves of Trend-Smooth training method.**

methods achieve higher test accuracies compared with the original methods.

C. EXPERIMENTAL ANALYSIS

To analyze the impact of Trend-Smooth on the training process, we randomly select 20 parameters from the last fully-connected layer of the neural network trained on CIFAR-10 dataset and record these parameters at each iteration. At the same time, we calculate the two indicators (i.e., Std Err and Values changed), which have been described in Section III, to see what Trend-Smooth has learned.

The parameter curves of each method are shown in Fig. 7. We also show the values of the indicators in Table 7. From the results we can see: 1) The curves of the A-SGD and A-SGD-alpha methods are more fluctuated than the SGD method and the Trend-Smooth method. Despite the lower learning rate, the A-SGD-alpha method's parameter curves are as fluctuated as the A-SGD method and value changes are almost

TABLE 6. Accuracies on CIFAR-10 dataset of adaptive learning rate methods and their Trend-Smooth counterparts.

Methods	Accuracies
A-Momentum	80.83%
A-RMSProp	82.27%
A-Adam	80.73%
Trend-Smooth+ A-Momentum	**81.57%**
Trend-Smooth+ A-RMSProp	**82.61%**
Trend-Smooth+ A-Adam	**81.92%**

TABLE 7. The summed standard error and values changed of these methods.

Methods	Std Err	Values Changed
A-SGD	5.81e-05	1.82
A-SGD-alpha	8.45e-05	1.93
A-DC	9.81e-05	1.83
SGD	3.89e-05	2.64
Trend-Smooth	$\mathbf{4.82e-05}$	$\mathbf{2.39}$

the same with A-SGD method. In comparison, the Trend-Smooth algorithm, which partially decreases the learning rate according to the paraemter trends, is more effective than the A-SGD method and the A-SGD-alpha method in reducing the fluctuation. Therefore, the Trend-Smooth method can get fast convergence and higher accuracy as shown previously. 2) The A-DC method's parameter curves have the largest fluctuation and the value changed is low. This is because that in the A-DC method's update rule, it adds a penalty to the gradient dimensions whose values accord with the parameter trends $((\mathbf{w}_{t+\tau} - \mathbf{w}_t))$. This is on the contrary to the Trend-Smooth method we propose. 3) The Trend-Smooth method's curves are much smoother compared to other asynchronous parallel methods and the parameters can change larger. Also its curves are more like the curves of SGD training method. Overall, our Trend-Smooth method, which leverages the parameter trend to adjust the learning rate, makes the parameter curves smoother and the variations larger. As a result, the Trend-Smooth method converges faster and obtains better performance.

Moreover, we analyze the overhead that Trend-Smooth adds on the system. The overhead is all on the server side since workers only compute gradients, send gradients and pull parameters. In each iteration, the main computing overhead comes from calculating the parameter trend on the server. In fact, it is quite easy to calculate the parameter trends, since it just needs to do a subtraction between two vectors. We could use the parallel utility provided by CPU to accelerate this job. Also, since for most parameters, the trend directions will not change during the training process, we could reduce the frequency of computing trends. As a result, Trend-Smooth's computation overhead is almost the same as compared to other asynchronous parallel methods. The storage overhead can be controlled by selecting the max length L of stack H. When the RAM is limited, the parameter copies can be stored on external storage (like disk).

VI. CONCLUSION

In this paper, we propose a novel algorithm called "Trend-Smooth" to accelerate the convergence speed and promote the test accuracy of the asynchronous parallel method. Our work is based on the analysis of the stale gradient impact on the parameter curves in the A-SGD method. To overcome the large fluctuation of the parameter curves in the A-SGD method, we leverage the parameter trend information to shrink the learning rate of certain gradient dimensions in order to make the parameter curves be smoother and change more like the SGD method. We have verified the effectiveness of the proposed method through experiments on MNIST and CIFAR-10 datasets. Experimental results confirm that Trend-Smooth could accelerate the training, get a lower loss value and achieve higher test accuracy (very close to the SGD method) than other asynchronous parallel methods. Trend-Smooth could also be combined with adaptive learning rate methods like Adam to further promote their performances in asynchronous parallel environment.

Finally, we emphasize that in this work we have focused on the application of the Trend-Smooth method in moderate neural network. In view of the mechanism of reducing the parameter fluctuations in our scheme, we expect this method can also work well for larger models, which we will investigate in the future work.

REFERENCES

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

[2] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[4] M. Li, D. G. Andersen, J. W. 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. OSDI*, Oct. 2014, vol. 1, p. 3.

[5] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.

[6] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2012, pp. 1223–1231.

[7] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 4120–4129.

[8] A. Sinha, M. Sarkar, A. Mukherjee, and B. Krishnamurthy, "Introspection: Accelerating neural network training by learning weight evolution," 2017, *arXiv:1704.04959*. [Online]. Available: https://arxiv.org/abs/1704.04959

[9] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2015, *arXiv:1603.04467*. [Online]. Available: https://arxiv.org/abs/1603.04467

[10] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.

[11] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful Quasi-Newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, Jul. 2009.

[12] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic Quasi-Newton method for online convex optimization," *Artif. Intell. Statist.*, pp. 436–443, Mar. 2007.

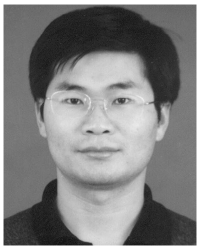
[13] A. Mokhtari and A. Ribeiro, "RES: Regularized stochastic BFGS algorithm," *IEEE Trans. Signal Process.*, vol. 62, no. 23, pp. 6089–6104, Dec. 2014.
[14] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory BFGS," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 3151–3181, Jan. 2015.
[15] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM J. Optim.*, vol. 19, no. 4, pp. 1574–1609, 2009.
[16] 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, Feb. 2011.
[17] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: https://arxiv.org/abs/1609.04747
[18] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4148–4158.
[19] L. Luo, W. Huang, Q. Zeng, Z. Nie, and X. Sun, "Learning personalized end-to-end goal-oriented dialog," 2018, *arXiv:1811.04604*. [Online]. Available: https://arxiv.org/abs/1811.04604
[20] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 3–19.
[21] W. Gropp, R. Thakur, and E. Lusk, *Using MPI–2—Advanced Features of the Message Passing Interface*. Cambridge, MA, USA: MIT Press, 1999.
[22] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, Sep. 1996.
[23] S. Owen and S. Owen, *Mahout in Action*. New York, NY, USA: Manning Shelter Island, 2012.
[24] T. White, *Hadoop: The Definitive Guide*. Newton, MA, USA: O'Reilly Media, Inc., 2012.
[25] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, Jan. 2016.
[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, Jun. 2010, p. 10.
[27] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[28] A. V. Gerbessiotis and L. G. Valiant, "Direct bulk-synchronous parallel algorithms," *J. Parallel Distrib. Comput.*, vol. 22, no. 2, pp. 251–267, 1994.
[29] T. Cheatham, A. Fahmy, D. Stefanescu, and L. Valiant, "Bulk synchronous parallel computing—A paradigm for transportable software," in *Tools and Environments for Parallel and Distributed Systems* Cham, Switzerland: Springer, 1996, pp. 61–76.
[30] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2014, pp. 571–582.
[31] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 285–322, Jan. 2015.
[32] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, Cambridge, MA, USA, vol. 1, 2015, pp. 685–693.
[33] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
[34] H. Avron, A. Druinsky, and A. Gupta, "Revisiting asynchronous linear solvers: Provable convergence rate through randomization," *J. ACM*, vol. 62, no. 6, Dec. 2015, Art. no. 51.
[35] B. McMahan and M. Streeter, "Delay-tolerant algorithms for asynchronous distributed online learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2915–2923.
[36] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 873–881.
[37] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2737–2745.
[38] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," 2015, *arXiv:1511.05950*. [Online]. Available: https://arxiv.org/abs/1511.05950
[39] X. Lian, H. Zhang, C.-J. Hsieh, Y. Huang, and J. Liu, "A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3054–3062.
[40] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in tensorflow," 2018, *arXiv:1802.05799*. [Online]. Available: https://arxiv.org/abs/1802.05799
[41] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, and D. Sreedhar, "PowerAI DDL," 2017, *arXiv:1708.02188*. [Online]. Available: https://arxiv.org/abs/1708.02188
[42] P. Goyal and P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv:1706.02677*. [Online]. Available: https://arxiv.org/abs/1706.02677
[43] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," 2018, *arXiv:1807.11205*. [Online]. Available: https://arxiv.org/abs/1807.11205
[44] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, vol. 1, no. 4.
[45] Y. LeCun, C. Cortes, and C. Burges. (Feb. 18, 2010). *Mnist Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist
[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

GUOXIN CUI is currently pursuing the Ph.D. degree with the CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences. His major research interest includes the training acceleration of machine learning models in distributed cluster.

JIAFENG GUO received the Ph.D. degree in computer software and theory from the University of Chinese Academy of Sciences, Beijing, China, in 2009. He has worked on a number of topics related to web search and data mining, including query representation and understanding, learning to rank, and text modeling. He is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences. He has published more than 80 articles in several top conferences/journals such as SIGIR, WWW, CIKM, IJCAI, and TKDE. His current research interests include representation learning and neural models for information retrieval and filtering. Moreover, he has served as the PC member for the prestigious conferences, including SIGIR, WWW, KDD, WSDM, and ACL. His work on information retrieval has received the Best Paper Award in ACM CIKM, in 2011, Best Student Paper Award in ACM SIGIR, in 2012, and Best Full Paper Runner-up Award in ACM CIKM, in 2017. He has served as an Associate Editor of TOIS.

YIXING FAN is currently an Assistant Professor with the Institute of Computing Technology, Chinese Academy of Sciences (ICT-CAS). His major research interests include information retrieval and machine learning. His research articles have been published in international conference, such as SIGIR, ACL, AAAI, WSDM, and CIKM. In CIKM 2017, he has received the Best Full Paper Runner-up Award. He has also lead an open source project for text matching (i.e., MatchZoo), which has been widely used by researchers.

YANYAN LAN is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences. She leads a research group working on Big Data and Machine Learning. Her current research interests include machine learning, web search and data mining, and big data analysis. She has published more than 60 papers on top conferences, including ICML, NIPS, SIGIR, and WWW, and the article entitled ''Top-k Learning to Rank: Labeling, Ranking, and Evaluation'' has received the Best Student Paper Award of SIGIR 2012. She was awarded Best Paper Runner-Up Award of CIKM, in 2017, Outstanding Reviewer of SIGIR, in 2017, and Youth Innovation Promotion Association, CAS.



XUEQI CHENG is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences. He is also a Ph.D. advisor, the Deputy Director, and the Director of Key Laboratory of Web Science and Technology, CAS. His main research areas include web search and data mining, data science, and social media analytics. He has more than 100 publications, and was awarded the Best Paper Award in ACM CIKM'11, and the Best Student Paper Award in ACM SIGIR'12. He is an Editorial Board Member of the *Journal of Computer Science and Technology* and the *Chinese Journal of Computer*. He is the General Secretary of CCF Task Force on Big Data, the Vice-Chair of CIPS Task Force on Chinese Information Retrieval. He was the General Co-Chair of WSDM'15, a Steering Committee Co-Chair of the IEEE Conference on Big Data, a PC Chair of ChinaCom'12 and IS'09, and PC member of more than 20 conferences, including ACM SIGIR, WWW, ACM CIKM, ACL, the IEEE ICDM, IJCAI, and ACM WSDM. He is an Associate Editor of the IEEE Transactions on Big Data.