On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization

Hao Yu [1] **Rong Jin** [1] **Sen Yang** [1]

Abstract

Recent developments on large-scale distributed machine learning applications, e.g., deep neural networks, benefit enormously from the advances in distributed non-convex optimization techniques, e.g., distributed Stochastic Gradient Descent (SGD). A series of recent works study the linear speedup property of distributed SGD variants with reduced communication. The linear speedup property enables us to scale out the computing capability by adding more computing nodes into our system. The reduced communication complexity is desirable since communication overhead is often the performance bottleneck in distributed systems. Recently, momentum methods are more and more widely adopted by practitioners to train machine learning models since they can often converge faster and generalize better. However, it remains unclear whether any distributed momentum SGD possesses the same linear speedup property as distributed SGD and has reduced communication complexity. This paper fills the gap by considering a distributed communication efficient momentum SGD method and proving its linear speedup property.

1. Introduction

Consider distributed non-convex optimization scenarios where N workers jointly solve the following consensus optimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^m} \quad f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\mathbb{E}_{\xi_i\sim\mathcal{D}_i}[F_i(\mathbf{x};\xi_i)]}_{\triangleq f_i(\mathbf{x})} \tag{1}$$

where $f_i(\mathbf{x}) \triangleq \mathbb{E}_{\xi_i\sim\mathcal{D}_i}[F_i(\mathbf{x};\xi_i)]$ are smooth non-convex functions with possibly different distributions $\mathcal{D}_i$. Problem (1) is particularly important in deep learning since it captures data parallelism for training deep neural networks. In deep learning with data parallelism, each $F_i(\cdot;\cdot) = F(\cdot;\cdot)$ represents the common deep neural network to be jointly trained and each $\mathcal{D}_i$ represents the distribution of the local data set accessed by worker i. The scenario where each $\mathcal{D}_i$ are different also captures the **federated learning** setting recently proposed in (McMahan et al., 2017) where mobile clients with private local training data and slow intermittent network connections cooperatively train a high-quality centralized model.

Stochastic Gradient Descent (SGD) (and its momentum variants) have been the dominating methodology for solving machine learning problem. For large-scale distributed machine learning problems, such as training deep neural networks, a parallel version of SGD, known as parallel mini-batch SGD, is widely adopted (Dean et al., 2012; Dekel et al., 2012; Li et al., 2014). With N parallel workers, parallel mini-batch SGD can solve problem (1) with $O(1/\sqrt{NT})$ convergence, which is N times faster[1] than the $O(1/\sqrt{T})$ convergence attained by SGD over a single node (Ghadimi & Lan, 2013; Lian et al., 2015). Obviously, such linear speedup with respect to (w.r.t.) number of workers is desired in distributed training as it implies perfect computation scalability by increasing the number of used workers. However, such linear speedup is difficult to harvest in practice because the classical parallel mini-batch SGD requires all workers to synchronize local gradients or models at **every** iteration such that inter-node communication cost becomes the performance bottleneck. To eliminate potential communication bottlenecks, such as high latency or low bandwidths, in computing infrastructures, many distributed SGD variants have been recently proposed. For example, (Lian et al., 2017; Jiang et al., 2017; Assran et al., 2018) consider decentralized parallel SGD where global gradient aggregations

[1]Machine Intelligence Technology Lab, Alibaba Group (U.S.) Inc., Bellevue, WA. Correspondence to: Hao Yu <eeyuhao@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]If an algorithm has $O(1/\sqrt{T})$ convergence, then it takes $1/\epsilon^2$ iterations to attain an $O(\epsilon)$ accurate solution. Similarly, if another algorithm has $O(1/\sqrt{NT})$ convergence, then it takes $1/(N\epsilon^2)$ iterations, which is N times smaller than $1/\epsilon^2$, to attain an $O(\epsilon)$ accurate solution. In this sense, the second algorithm is N times faster than the first one.

used in the classical parallel mini-batch SGD are replaced with local aggregations between neighbors. To reduce the communication cost at each iteration, compression or sparsification based parallel SGD are considered in (Seide et al., 2014; Strom, 2015; Aji & Heafield, 2017; Wen et al., 2017; Alistarh et al., 2017). Recently, (Yu et al., 2018; Wang & Joshi, 2018a; Jiang & Agrawal, 2018) prove that certain parallel SGD variants that strategically skip communication rounds can achieve the fast $O(1/\sqrt{NT})$ convergence with significantly less communication rounds. See Table 1 for a summary on recent works studying distributed SGD with $O(1/\sqrt{NT})$ convergence and reduced communication complexity.

It is worth noting that existing convergence analyses on distributed methods for solving problem (1) focus on parallel SGD **without momentum**. In practice, momentum SGD is more commonly used to train deep neural networks since it often can converge faster and generalize better (Krizhevsky et al., 2012; Sutskever et al., 2013; Yan et al., 2018). For example, momentum SGD is suggested for training ResNet for image classification tasks to obtain the best test accuracy (He et al., 2016). See Figure 1 for the comparison of test accuracy between training with momentum and training without momentum.[2] In fact, while previous works (Lian et al., 2017; Stich, 2018; Yu et al., 2018; Jiang & Agrawal, 2018) prove that distributed vanilla SGD (without momentum) can train deep neural networks with $O(1/\sqrt{NT})$ convergence using significantly fewer communications rounds, most of their experiments use momentum SGD rather than vanilla SGD (to achieve the targeted test accuracy). In addition, previous empirical works (Chen & Huo, 2016; Lin et al., 2018) on distributed training for deep neural networks explicitly observe that momentum is necessary to improve the convergence and test accuracy. In this perspective, there is a huge gap between the current practices, i.e., using momentum SGD rather than vanilla SGD in distributed training for deep neural networks, and existing theoretical analyses such as (Yu et al., 2018; Wang & Joshi, 2018a; Jiang & Agrawal, 2018) studying the convergence rate and communication complexity of SGD without momentum. Momentum methods are originally proposed by Polyak in (Polyak, 1964) to minimize deterministic strongly convex quadratic functions. Its convergence rate for deterministic convex optimization, which is not necessarily strongly convex, is established in (Ghadimi et al., 2014). For non-convex optimization, the convergence for deterministic non-convex optimization is proven in (Zavriev & Kostyuk, 1993) and the $O(1/\sqrt{T})$ convergence rate for stochastic non-convex optimization is recently shown in (Yan et al., 2018). However, to our best knowledge, it remains as an open question: "*Whether any distributed momentum SGD can achieve the same $O(1/\sqrt{NT})$ convergence, i.e., linear speedup w.r.t. the number of workers, with reduced communication complexity as SGD (without momentum) methods in (Yu et al., 2018; Wang & Joshi, 2018a; Jiang & Agrawal, 2018) ?*"

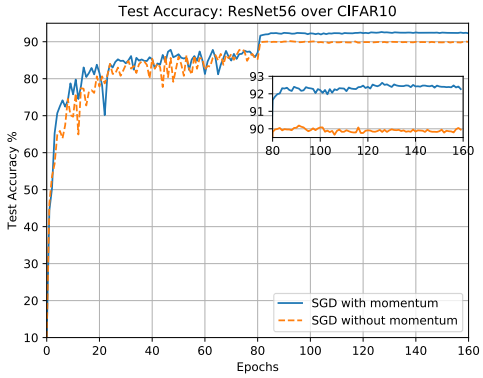


Figure 1: Test accuracy performance of ResNet56 over CIFAR10 when trained with SGD with and without momentum on a single GPU using hyper-parameters suggested in (He et al., 2016).

Our Contributions: In this paper, we considers parallel restarted SGD with momentum (described in Algorithm 1), which can be viewed as the momentum extension of parallel restarted SGD, also referred as local SGD, considered in (Stich, 2018; Yu et al., 2018). Such a method is also faithful to the common practice "model averaging with momentum" used by practitioners for training deep neural networks in (Chen & Huo, 2016; McMahan et al., 2017; Su et al., 2018; Lin et al., 2018). We show that parallel restarted SGD with momentum can solve problem (1) with $O(1/\sqrt{NT})$ convergence, i.e., achieving linear speedup w.r.t. number of workers. Moreover, to achieve the fast $O(1/\sqrt{NT})$ convergence, T iterations of our algorithm only requires $O(N^{3/2}T^{1/2})$ communication rounds when all workers can access identical data sets; or $O(N^{3/4}T^{3/4})$ communication rounds when workers access non-identical data sets. To our knowledge, this is the first time that a distributed momentum SGD method for non-convex stochastic optimization is proven to possess the same linear speedup property (with communication reduction) as distributed SGD (without momentum) in (Lian et al., 2017; Yu et al., 2018; Wang & Joshi, 2018a; Jiang & Agrawal, 2018).

Recall that momentum SGD degrades to vanilla SGD when its momentum coefficient is set 0. Algorithm 1 with $\beta = 0$ degrades to the parallel SGD methods (without momentum) considered in (Yu et al., 2018; Wang & Joshi, 2018a;

[2] As observed in Figure 1, the final test accuracy of momentum SGD is roughly 2.5% better than that of vanilla SGD (without momentum) over CIFAR10. In practice, people usually use momentum SGD to train ResNet in both single GPU and multiple GPU scenarios. Some practitioners conjecture that it is possible to avoid the performance degradation of vanilla SGD if its hyper-parameters are well tuned. However, even if this conjecture is true, the hyper-parameter tuning can be extremely time-consuming.

Table 1: Number of communication rounds involved in T iterations of existing distributed SGD **without momentum** with $O(1/\sqrt{NT})$ convergence for **non-convex** stochastic optimization (1).

REFERENCE	IDENTICAL $f_i(\mathbf{x})$, I.E.,$\kappa = 0$	NON-IDENTICAL $f_i(\mathbf{x})$, I.E., $\kappa \neq 0$	EXTRA ASSUMPTIONS
(LIAN ET AL., 2017)	$O(T)$	$O(T)$	NO
(YU ET AL., 2018)	$O(N^{3/4}T^{3/4})$	$O(N^{3/4}T^{3/4})$	BOUNDED GRADIENTS
(JIANG & AGRAWAL, 2018)	$O(N^{5/2}T^{1/2})$	$O(N^{5/4}T^{3/4})$	NO
(WANG & JOSHI, 2018A)	$O(N^{3/2}T^{1/2})$	NOT APPLICABLE	NO
THIS PAPER	$O(N^{3/2}T^{1/2})$	$O(N^{3/4}T^{3/4})$	NO

Jiang & Agrawal, 2018). Our communication complexity results for parallel SGD without momentum cases also improve the state-of-the-art. As shown in Table 1, the number of required communication rounds shown in this paper is the fewest in both identical training data set case and non-identical data set case. In particular, this paper relaxes the bounded gradient moment assumption used in (Yu et al., 2018) to a milder bounded variance assumption and reduces the communication complexity for the case with identical training data. Our analysis applies to the distributed training scenarios where workers access non-identical training sets, e.g., training with sharded data or federated learning, that can not be handled in (Wang & Joshi, 2018a).

This paper further considers momentum SGD with decentralized communication and proves that it possess the same linear speedup property as its vanilla SGD (without momentum) counterpart considered in (Lian et al., 2017).

2. Parallel Restarted SGD with Momentum

2.1. Preliminary

Following the convention in (distributed) stochastic optimization, we assume each worker can independently evaluate unbiased stochastic gradients $\nabla F_i(\mathbf{x}_i;\xi_i), \xi_i \sim \mathcal{D}_i$ using its own local variable $\mathbf{x}_i$. Throughout this paper, we have the following standard assumption:

Assumption 1. *Problem* (1) *satisfies the following:*

1. **Smoothness:** *Each function $f_i(\mathbf{x})$ in problem* (1) *is smooth with modulus L.*
2. **Bounded variances:** *There exist $\sigma > 0$ and $\kappa > 0$ such that*

$$\mathbb{E}_{\xi\sim\mathcal{D}_i}[\|\nabla F_i(\mathbf{x};\xi) - \nabla f_i(\mathbf{x})\|^2] \leq \sigma^2, \forall i, \forall \mathbf{x} \quad (2)$$

$$\frac{1}{N}\sum_{i=1}^{N}\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \kappa^2, \forall \mathbf{x} \quad (3)$$

Note that σ^2 in Assumption 1 quantifies the variance of stochastic gradients at local worker, and κ^2 quantifies the deviations between each worker's local objective function $f_i(\mathbf{x})$. For distributed training of neural networks, if all workers can access the same data set, then $\kappa = 0$. Assumption 1 was previously used in (Lian et al., 2017; Wen et al., 2017; Alistarh et al., 2017; Jiang & Agrawal, 2018). The bounded variance assumption in Assumption 1 is milder than the bounded second moment assumption used in (Stich, 2018; Yu et al., 2018).

Recall that if function $f(\cdot)$ is smooth with modulus L, then we have the key property that $f(\mathbf{x}) \leq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y}\rangle + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2, \forall \mathbf{x}, \mathbf{y}$ where $\langle\cdot,\cdot\rangle$ represents the inner product of two vectors. Another two useful properties implied by Assumption 1 are summarized in the next two lemmas.

Lemma 1. *Consider problem* (1) *under Assumption 1. Let $\mathbf{g}_i$ be mutually independent unbiased stochastic gradients sampled at points $\mathbf{x}_i$ such that $\mathbb{E}[\mathbf{g}_i] = \nabla f_i(\mathbf{x}_i), \forall i \in \{1, 2, \ldots, N\}$. Then, we have*

$$\mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i\|^2] \leq \frac{1}{N}\sigma^2 + \mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{x}_i)\|^2]$$

Proof. See Supplement 6.1 or our arXiv full version. □

Since $\mathbb{E}[\mathbf{g}_i] = \nabla f_i(\mathbf{x}_i)$, we have $\mathbb{E}[\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i] = \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{x}_i)$. The implication of Lemma 1 is: while $\mathbf{g}_i$ are sampled at different points $\mathbf{x}_i$, the variance of $\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i$, which is equal to $\mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i - \frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{x}_i)\|^2] = \mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i\|^2] - \mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\nabla f_i(\mathbf{x}_i)\|^2]$, scales down by N times when N workers sample gradients independently (even though at different $\mathbf{x}_i$). Note that if each $\mathbf{g}_i$ are independently sampled by N workers at the same point $\mathbf{x}_i \equiv \mathbf{x}, \forall i$ (as in the classical parallel mini-batch SGD), then Lemma 1 reduces to $\mathbb{E}[\|\frac{1}{N}\sum_{i=1}^{N}\mathbf{g}_i - \nabla f(\mathbf{x})\|^2] \leq \frac{\sigma^2}{N}$, which is the fundamental reason why the classical parallel mini-batch SGD can converges N times faster with N workers. However, to reduce the communication overhead associated with synchronization between workers, we shall consider distributed algorithms with fewer synchronization rounds such that $\mathbf{x}_i$ at different workers can possibly deviate from

each other. In this case, huge deviations across $\mathbf{x}_i$ can cause that $\mathbf{g}_i$ sampled at $\mathbf{x}_i$ provides unreliable or even contradicting gradient knowledge and hence slow down the convergence. Thus, to let N workers jointly solve problem (1) with fast convergence, the distributed algorithm needs to enforce certain concentration of $\mathbf{g}_i$. The following useful lemma relates the concentration of $\nabla f_i(\mathbf{x})$ with deviations across all $\mathbf{x}_i$.

Lemma 2. *Consider problem* (1) *under Assumption 1. For any N points $\mathbf{x}_i, i \in \{1, 2, \ldots, N\}$, if we define $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i$,then we have*

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^N \|\nabla f_i(\mathbf{x}_i) - \frac{1}{N}\sum_{j=1}^N \nabla f_j(\mathbf{x}_j)\|^2\\ \leq&\frac{6L^2}{N}\sum_{i=1}^N \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 + \frac{3}{N}\sum_{i=1}^N \|\nabla f_i(\bar{\mathbf{x}}) - \nabla f(\bar{\mathbf{x}})\|^2\end{aligned}$$

Proof. See Supplement 6.2 or our arXiv full version. □

2.2. Parallel Restarted SGD with Momentum

2.2.1. Algorithm 1 with Polyak's Momentum

Consider applying the distributed algorithm described in Algorithm 1 to solve problem (1) with N workers. In the literature, there are two momentum methods for SGD, i.e., Polyak's momentum method (also known as the heavy ball method) and Nesterov's momentum method. Algorithm 1 can use either of them for local worker updates by providing two options: "Option I" is Polyak's momentum; "Option II" is Nesterov's momentum.[3] We also note that the update steps described in (4) or (5) can be locally performed at each worker in parallel.

The synchronization step (6) can be interpreted as restarting momentum SGD with **new** initial point $\hat{\mathbf{x}}$ and momentum buffer $\hat{\mathbf{u}}$, i.e., resetting $\mathbf{u}_i^{(t)} = \hat{\mathbf{u}}$ and $\mathbf{x}_i^{(t)} = \hat{\mathbf{x}}$. In this perspective, we call Algorithm 1 "parallel restarted SGD with momentum". Note that if we use $\beta = 0$ in Algorithm 1, then it degrades to the "parallel restarted SGD", also known as "local SGD" or "SGD with periodic averaging", in (Stich, 2018; Yu et al., 2018; Lin et al., 2018; Wang & Joshi, 2018a; Zhou & Cong, 2018; Jiang & Agrawal, 2018).

Since inter-node communication is only needed by Algorithm 1 to calculate global averages $\hat{\mathbf{x}}$ and $\hat{\mathbf{u}}$ in (6) and happens only once every I iterations, the total number of inter-communication rounds involved in T iterations of Algorithm 1 is given by T/I. If we use **all-reduce** operations to compute the global averages, the per-round communication is relatively cheap and does not involve a centralized parameter server (Goyal et al., 2017). Algorithm 1 is suitable to the **federated learning** scenario with high demand on privacy and security since workers exchange models rather than raw data or gradients.

[3]The current description in (4) and (5) is identical to the default implementations of momentum methods in PyTorch's SGD optimizer. Polyak's and Nesterov's momentum have many other equivalent representations. These equivalent representations will be further discussed later.

Algorithm 1 Parallel Restarted SGD with momentum (PR-SGD-Momentum)

Input: Initialize local solutions $\mathbf{x}_i^{(0)} = \hat{\mathbf{x}} \in \mathbb{R}^m$ and momentum buffers $\mathbf{u}_i^{(0)} = \mathbf{0}, \forall i \in \{1, 2, \ldots, N\}$. Set learning rate $\gamma > 0$, momentum coefficient $\beta \in [0, 1)$, node synchronization interval $I > 0$ and number of iterations T

for $t = 1, 2 \ldots, T-1$ **do**

Each worker i samples its stochastic gradient $\mathbf{g}_i^{(t-1)} = \nabla F_i(\mathbf{x}_i^{(t-1)}; \xi_i^{(t-1)})$ with $\xi_i^{(t-1)} \sim \mathcal{D}_i$.

Each worker i in parallel updates its local momentum buffer and solution via

$$\text{Option I: } \begin{cases}\mathbf{u}_i^{(t)} = \beta\mathbf{u}_i^{(t-1)} + \mathbf{g}_i^{(t-1)}\\ \mathbf{x}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma\mathbf{u}_i^{(t)}\end{cases} \quad \forall i. \tag{4}$$

Or

$$\text{Option II: } \begin{cases}\mathbf{u}_i^{(t)} = \beta\mathbf{u}_i^{(t-1)} + \mathbf{g}_i^{(t-1)}\\ \mathbf{v}_i^{(t)} = \beta\mathbf{u}_i^{(t)} + \mathbf{g}_i^{(t-1)}\\ \mathbf{x}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma\mathbf{v}_i^{(t)}\end{cases} \quad \forall i. \tag{5}$$

if t is a multiple of I, i.e., $t \bmod I = 0$, **then**

Each worker resets its momentum buffer and local solution as the node averages via

$$\begin{cases}\mathbf{u}_i^{(t)} = \hat{\mathbf{u}} \triangleq \frac{1}{N}\sum_{j=1}^N \mathbf{u}_j^{(t)}\\ \mathbf{x}_i^{(t)} = \hat{\mathbf{x}} \triangleq \frac{1}{N}\sum_{j=1}^N \mathbf{x}_j^{(t)}\end{cases} \quad \forall i \tag{6}$$

end if

end for

Algorithm 1 uses $\{\mathbf{x}_i^{(t)}\}_{t\geq 0}$ to denote local solution sequences generated at each worker i. For each iteration t, we define

$$\bar{\mathbf{x}}^{(t)} \triangleq \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i^{(t)} \tag{7}$$

as the averages of local solutions $\mathbf{x}_i^{(t)}$ across all N nodes. Our performance analysis will be performed regarding the aggregated version $\bar{\mathbf{x}}^{(t)}$ defined in (7) so that "consensus" is no longer our concern for problem (1). Performing convergence analysis for the node-average version $\bar{\mathbf{x}}^{(t)}$ has been an important technique used in previous works on distributed consensus optimization (Lian et al., 2017; Mania et al., 2017;

Stich, 2018; Yu et al., 2018; Jiang & Agrawal, 2018).

2.3. Performance Analysis

This subsection analyzes the convergence rates of Algorithm 1 with Polyak's and Nesterov's momentum, respectively.

We first consider Algorithm 1 with Option I given by (4), i.e., Polyak's momentum. Polyak's momentum SGD, also known as the heavy ball method, is the classical momentum SGD used for training deep neural networks and often provides fast convergence and good generalization (Krizhevsky et al., 2012; Sutskever et al., 2013; Yan et al., 2018). If we let $\mathbf{x}_i^{(-1)} = \mathbf{x}_i^{(0)}$, then (4) in Algorithm 1 can be equivalently written as the following single variable version

$$\mathbf{x}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma \mathbf{g}_i^{(t-1)} + \beta[\mathbf{x}_i^{(t-1)} - \mathbf{x}_i^{(t-2)}] \quad (8)$$

where the last term $\beta[\mathbf{x}_i^{(t-1)} - \mathbf{x}_i^{(t-2)}]$ is often called Polyak's momentum term.

It's interesting to note that if we define an auxiliary sequence

$$\bar{\mathbf{u}}^{(t)} \triangleq \frac{1}{N}\sum_{i=1}^{N} \mathbf{u}_i^{(t)} \quad (9)$$

which is the node average sequence of local buffer variables $\mathbf{u}_i^{(t)}$ from Algorithm 1 with Option I, then we have

$$\begin{cases} \bar{\mathbf{u}}^{(t)} & = \beta \bar{\mathbf{u}}^{(t-1)} + \frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i^{(t-1)} \\ \bar{\mathbf{x}}^{(t)} & = \bar{\mathbf{x}}^{(t-1)} - \gamma \bar{\mathbf{u}}^{(t)} \end{cases}, \forall t \geq 1 \quad (10)$$

where $\bar{\mathbf{x}}^{(t)}$ is defined in (7).

An important observation is that if $\frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i^{(t-1)}$ in (10) is replaced with $\nabla f(\bar{\mathbf{x}}^{(t-1)})$, then (10) is exactly a standard **single-node** SGD momentum method with momentum buffer $\bar{\mathbf{u}}^{(t)}$ and solution $\bar{\mathbf{x}}^{(t)}$. That is, under Algorithm 1, N workers essentially jointly update $\bar{\mathbf{u}}^{(t)}$ and $\bar{\mathbf{x}}^{(t)}$ with momentum SGD using an **inaccurate** stochastic gradient $\frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i^{(t-1)}$. However, since (6) periodically (every I iterations) synchronizes all local variables $\mathbf{u}_i^{(t)}$ and $\mathbf{x}_i^{(t)}$, our intuition is by synchronizing frequently enough the inaccuracy of the used gradient counterpart in (10) shall not damage the convergence too much. The next theorem summarizes the convergence rate of Algorithm 1 and characterizes the effect of synchronization interval I in its convergence.

Remark 1. *Our Algorithm 1 is different from a common heuristic model averaging strategy for momentum SGD suggested in (Wang & Joshi, 2018b) and in Microsoft's CNTK framework (Seide & Agarwal, 2016). The strategy in (Seide & Agarwal, 2016; Wang & Joshi, 2018b) let each worker run local momentum SGD in parallel, and periodically reset momentum buffer variables to zero and average local models. However, there is no convergence analysis for this strategy. In contrast, this paper shall provide rigorous convergence analysis for our Algorithm 1. Our experiment in Supplement 6.7.2 furthers shows that Algorithm 1 has faster convergence than the strategy in (Seide & Agarwal, 2016; Wang & Joshi, 2018b) and yields better test accuracy when used to train ResNet for CIFAR10.*

Theorem 1. *Consider problem* (1) *under Assumption 1. If we use* $\gamma \leq \frac{(1-\beta)^2}{(1+\beta)L}$ *and* $I \leq \frac{1-\beta}{6L\gamma}$ *in Algorithm 1 with* **Option I**, *then for all* $T \geq 1$, *we have*

$$\begin{aligned} &\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] \\ \leq &\frac{2(1-\beta)}{\gamma T}(f(\bar{\mathbf{x}}^{(0)}) - f^*) + \frac{L\gamma}{(1-\beta)^2}\frac{\sigma^2}{N} + \frac{4L^2\gamma^2 I\sigma^2}{(1-\beta)^2} + \frac{9L^2\gamma^2 I^2\kappa^2}{(1-\beta)^2} \\ = &O(\frac{1}{\gamma T}) + O(\frac{\gamma}{N}\sigma^2) + O(\gamma^2 I\sigma^2) + O(\gamma^2 I^2\kappa^2) \end{aligned}$$

where $\{\bar{\mathbf{x}}^{(t)}\}_{t\geq 0}$ *is the sequence defined in* (7)*; and* σ *and* κ *are constants defined in Assumption 1; and* f^* *is the minimum value of problem* (1).

Proof. See Supplement 6.3 or our arXiv full version. □

The next corollary summarizes that Algorithm 1 with Option I using N workers can solve problem (1) with the fast $O(\frac{1}{\sqrt{NT}})$ convergence, i.e., achieving the linear speedup (w.r.t. number of workers). Note that $O(\frac{1}{\sqrt{NT}})$ dominates $O(\frac{N}{T})$ in Corollary 1 when T is sufficiently large.

Corollary 1 (Linear Speedup)**.** *Consider problem* (1) *under Assumption 1. If we use* $\gamma = \frac{\sqrt{N}}{\sqrt{T}}$ *and* $I = 1$ *in Algorithm 1 with* **Option I**, *then for any* $T \geq \frac{36L^2N}{(1-\beta)^2}$, *we have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] = O(\frac{1}{\sqrt{NT}}) + O(\frac{N}{T}). \quad (11)$$

Proof. This corollary follows because the selection of γ and I satisfies the conditions in Theorem 1. □

The next corollary summarizes that the desired linear speedup can be attained with communication skipping, i.e., using $I > 1$ in Algorithm 1. In particular, to achieve the linear speedup, T iterations in Algorithm 1 with Option I only require $O(N^{3/2}T^{1/2})$ communication rounds when $\kappa = 0$, i.e., workers access the common training data set; or only require $O(N^{3/4}T^{3/4})$ communication rounds when $\kappa \neq 0$, i.e., workers access non-identical training data sets.

Corollary 2 (Linear Speedup with Reduced Communication)**.** *Consider problem* (1) *under Assumption 1.*

- **Case** $\kappa = 0$*: If we use* $\gamma = \frac{\sqrt{N}}{\sqrt{T}}$ *and* $I \leq \frac{1-\beta}{6L}\frac{\sqrt{T}}{N^{3/2}} = O(\frac{T^{1/2}}{N^{3/2}})$ *in Algorithm 1 with* **Option I**, *then for any* $T \geq \frac{(1+\beta)^2L^2}{(1-\beta)^4}N$, *we have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] = O(\frac{1}{\sqrt{NT}}). \quad (12)$$

By using $I = O(\frac{T^{1/2}}{N^{3/2}})$, the total number of inter-node communication rounds involved in Algorithm 1 is $O(N^{3/2}T^{1/2})$.

- **Case $\kappa \neq 0$:** *If we use $\gamma = \frac{\sqrt{N}}{\sqrt{T}}$ and $I \leq \frac{1-\beta}{6L}\frac{T^{1/4}}{N^{3/4}} = O(\frac{T^{1/4}}{N^{3/4}})$ in Algorithm 1 with* **Option I**, *then for any $T \geq \frac{(1+\beta)^2L^2}{(1-\beta)^4}N$, we have*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] = O(\frac{1}{\sqrt{NT}}). \tag{13}$$

By using $I = O(\frac{T^{1/4}}{N^{3/4}})$, the total number of inter-node communication rounds involved in Algorithm 1 is $O(N^{3/4}T^{3/4})$.

Proof. This corollary follows simply because the selection of γ and I satisfies the conditions in Theorem 1. □

Remark 2. *Following the convention in non-convex optimization, the convergence rate in Corollary 2 is measured by the expected squared gradient norm used in (Ghadimi & Lan, 2013; Lian et al., 2017; Yu et al., 2018; Jiang & Agrawal, 2018). To attain the average $\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2]$ in expectation, one neat strategy suggested in (Ghadimi & Lan, 2013) is to generate a random iteration count R uniformly from $\{0, 1, \ldots, T-1\}$, terminate Algorithm 1 at iteration R and output $\bar{\mathbf{x}}^{(R)}$ as the solution.*

Remark 3. *Note that Theorem 1 and Corollary 2 hold for any $0 \leq \beta < 1$. Recall that Algorithm 1 with $\beta = 0$ degrades to parallel restarted SGD (without momentum). For parallel SGD (without momentum), the communication complexity implied by Corollary 2 improves the state-of-the-art as summarized in Table 1.*

2.3.1. Algorithm 1 with Nesterov's Momentum

Now consider using Nesterov's momentum described in Option II in Algorithm 1. Option II given by (5) introduces extra auxiliary variables $\bar{\mathbf{v}}_i^{(t)}$ and uses them in the update of local solutions $\mathbf{x}_i^{(t)}$. It is not difficult[4] to show that (5) yields the same solution sequences $\{\mathbf{x}_i^{(t)}\}_{t\geq 0}$ as

$$\begin{cases}\mathbf{y}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma \mathbf{g}_i^{(t-1)} \\ \mathbf{x}_i^{(t)} = \mathbf{y}_i^{(t)} + \beta[\mathbf{y}_i^{(t)} - \mathbf{y}_i^{(t-1)}]\end{cases} \quad \forall i \tag{14}$$

with $\mathbf{y}_i^{(0)} \triangleq \mathbf{0}, \forall i$. The only difference between (5) and (14) is the adoption of different cache variables.

Equation (14) is more widely used to describe SGD with Nesterov's momentum in the literature (Nesterov, 2004). However, by writing Nesterov's momentum SGD as (5), an important observation is that momentum buffer variables $\mathbf{u}_i^{(t)}$ in Polyak's and Nesterov's momentum methods evolve according to the same dynamic (with stochastic gradients sampled at different points). By adapting the convergence rate analysis for Polyak's momentum, Theorem 2 (formally proven in Supplement 6.5) summarizes the convergence for Nesterov's momentum.

Theorem 2. *Consider problem* (1) *under Assumption 1. If we use $\gamma \leq \frac{(1-\beta)^2}{L(1+\beta^3)}$ and $I \leq \frac{1-\beta}{6L\gamma}$ in Algorithm 1 with* **Option II**, *then for all $T > 0$, we have*

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] \\ \leq&\frac{2(1-\beta)}{\gamma T}(f(\bar{\mathbf{x}}^{(0)}) - f^*) + \frac{L\gamma}{(1-\beta)^2}\frac{\sigma^2}{N} + \frac{4L^2\gamma^2 I\sigma^2}{(1-\beta)^2} + \frac{9L^2\gamma^2 I^2\kappa^2}{(1-\beta)^2} \\ =&O(\frac{1}{\gamma T}) + O(\frac{\gamma}{N}\sigma^2) + O(\gamma^2 I\sigma^2) + O(\gamma^2 I^2\kappa^2)\end{aligned}$$

where $\{\bar{\mathbf{x}}^{(t)}\}_{t\geq 0}$ is the sequence defined in (7)*; and σ and κ are constants defined in Assumption 1; and f^* is the minimum value of problem* (1).

Remark 4. *By comparing Theorem 1 and Theorem 2, we conclude that the order of convergence rates for both options in Algorithm 1 (with slightly different choices of learning rate γ) have the same dependence on γ, N and I. It is then immediate that Algorithm 1 with Option II can achieve the linear speedup with the same (reduced) communication complexity as summarized in Corollary 2 for Option I.*

3. Extension: Momentum SGD with Decentralized Communication

Note that Algorithm 1 requires to compute global averages of all N nodes at each communication step (6). Such global averages are possible without a central parameter center by using distributive average protocols such as all-reduce primitives (Goyal et al., 2017). However, the feasibility and performance of exiting all-reduce protocols are restricted to the underlying network topology. For example, the performance of a ring based all-reduce protocol can be poor in a line network, which does not have a physical ring. In this subsection, we consider distributed momentum SGD with decentralized communication (described in Algorithm 2) where the communication pattern can be fully customized. Let $\mathbf{W} \in \mathbb{R}^{N\times N}$ be a symmetric doubly stochastic matrix. By definition, a symmetric doubly stochastic matrix satisfies (1) $W_{ij} \in [0, 1], \forall i, j \in \{1, 2, \ldots, N\}$; (2) $\mathbf{W}^{\mathsf{T}} = \mathbf{W}$; (3) $\sum_{j=1}^{N} W_{ij} = 1, \forall i \in \{1, 2, \ldots, N\}$. For a computer network with N nodes, we can use $\mathbf{W}$ to encode the communication between nodes, i.e., if node i and node j are disconnected, we let $W_{ij} = 0$. This section further assumes the mixing matrix $\mathbf{W}$ that obeys the network connection topology is selected to satisfy the following assumption. The same assumption is used in (Lian et al., 2017; Jiang et al., 2017; Wang & Joshi, 2018a).

[4] The derivation on the equivalence is in Supplement 6.4.

Algorithm 2 Distributed Momentum SGD with Decentralized Communication

1: **Input:** Initialize local solutions $\mathbf{x}_i^{(0)} = \hat{\mathbf{x}} \in \mathbb{R}^m$ and momentum buffers $\mathbf{u}_i^{(0)} = \mathbf{0}, \forall i \in \{1, 2, \ldots, N\}$. Set mixing matrix $\mathbf{W}$, learning rate $\gamma > 0$, momentum coefficient $\beta \in [0, 1)$ and number of iterations T

2: **for** $t = 1, 2 \ldots, T-1$ **do**

3: Each worker i samples its stochastic gradient $\mathbf{g}_i^{(t-1)} = \nabla F_i(\mathbf{x}_i^{(t-1)}; \xi_i^{(t-1)})$ with $\xi_i^{(t-1)} \sim \mathcal{D}_i$.

4: Each worker i in parallel updates its local auxiliary momentum buffer $\tilde{\mathbf{u}}_i^{(t)}$ and solution $\tilde{\mathbf{x}}_i^{(t)}$ via

$$\text{Option I:} \begin{cases} \tilde{\mathbf{u}}_i^{(t)} = \beta \mathbf{u}_i^{(t-1)} + \mathbf{g}_i^{(t-1)} \\ \tilde{\mathbf{x}}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma \tilde{\mathbf{u}}_i^{(t)} \end{cases} \quad \forall i. \quad (16)$$

Or

$$\text{Option II:} \begin{cases} \tilde{\mathbf{u}}_i^{(t)} = \beta \mathbf{u}_i^{(t-1)} + \mathbf{g}_i^{(t-1)} \\ \tilde{\mathbf{v}}_i^{(t)} = \beta \tilde{\mathbf{u}}_i^{(t)} + \mathbf{g}_i^{(t-1)} \\ \tilde{\mathbf{x}}_i^{(t)} = \mathbf{x}_i^{(t-1)} - \gamma \tilde{\mathbf{v}}_i^{(t)} \end{cases} \quad \forall i. \quad (17)$$

5: Each worker i updates its local momentum buffer $\mathbf{u}_i^{(t)}$ and solution $\mathbf{x}_i^{(t)}$ based on the neighborhood weighted averages given by

$$\begin{cases} \mathbf{u}_i^{(t)} = \sum_{j=1}^N \tilde{\mathbf{u}}_j^{(t)} W_{ji} \\ \mathbf{x}_i^{(t)} = \sum_{j=1}^N \tilde{\mathbf{x}}_j^{(t)} W_{ji} \end{cases} \quad \forall i \quad (18)$$

6: **end for**

Assumption 2. *The mixing matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ is a symmetric doubly stochastic matrix satisfying $\lambda_1(\mathbf{W}) = 1$ and*

$$\max\{|\lambda_2(\mathbf{W})|, |\lambda_N(\mathbf{W})|\} \leq \sqrt{\rho} < 1 \quad (15)$$

where $\lambda_i(\mathbf{W})$ is the i-th largest eigenvalue of $\mathbf{W}$.

Compared with PR-SGD-Momentum described in Algorithm 1, Algorithm 2 uses a **local** aggregation step (18) at every iteration. While Algorithm 2 can not skip the aggregation steps as Algorithm 1 does and hence involves I times more communication rounds, the per-step communication is more flexible since each node only computes neighborhood averages rather than global averages. As a consequence, Algorithm 2 is more suitable to distributed machine learning with mobiles where the network topology is heterogeneous and diverse. By extending our analysis for Algorithm 1, the next theorem proves the linear speedup of Algorithm 2.

Theorem 3. *Consider problem* (1) *under Assumptions 1 and 2. If we use $\gamma \leq \min\{\frac{(1-\beta)^2(1-\sqrt{\rho})^2}{6L}, \frac{(1-\beta)(1-\sqrt{\rho})}{4L}\}$ in Algorithm 2, then for all $T \geq 1$, we have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] = O(\frac{1}{\gamma T}) + O(\frac{\gamma}{N}\sigma^2) + O(\gamma^2\sigma^2) + O(\gamma^2\kappa^2)$$

where $\{\bar{\mathbf{x}}^{(t)}\}_{t\geq 0}$ defined in (7) *are the averages of local solutions across N workers; and σ, κ and ρ are constants defined in Assumptions 1-2.*

Proof. See Supplement 6.6 or our arXiv full version. □

This next corollary further summarizes that using $\gamma = O(\frac{\sqrt{N}}{\sqrt{T}})$ in Algorithm 2 can achieve $O(\frac{1}{\sqrt{NT}})$ convergence with the linear speedup.

Corollary 3. *Consider problem* (1) *under Assumptions 1 and 2. If we use $\gamma = \frac{\sqrt{N}}{\sqrt{T}}$ in Algorithm 2, then for all $T \geq \max\{\frac{36NL^2}{(1-\beta)^4(1-\sqrt{\rho})^4}, \frac{32NL^2}{(1-\beta)^2(1-\sqrt{\rho})^2}\}$, we have*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\bar{\mathbf{x}}^{(t)})\|^2] = O(\frac{1}{\sqrt{NT}}) + O(\frac{N}{T})$$

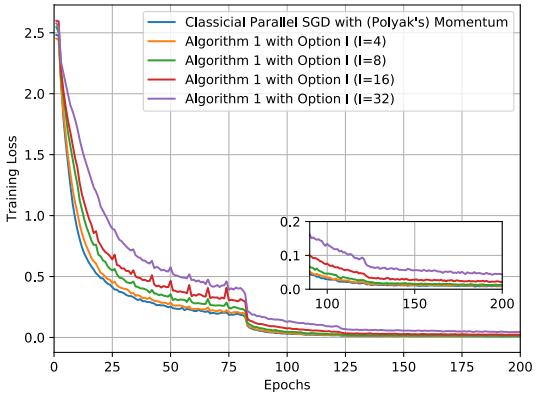
where $\{\bar{\mathbf{x}}^{(t)}\}_{t\geq 0}$ defined in (7) *are the averages of local solutions across N workers.*

Note that Algorithm 2 with $\beta = 0$ degrades to the decentralized SGD without momentum considered in (Lian et al., 2017), where the linear speedup of decentralized SGD is first shown. Recently, many variants of decentralized SGD without momentum with linear speedup have been developed for distributed deep learning (Tang et al., 2018; Wang & Joshi, 2018a; Assran et al., 2018). To our knowledge, this is the first time that decentralized momentum SGD is shown to possess the same linear speedup.

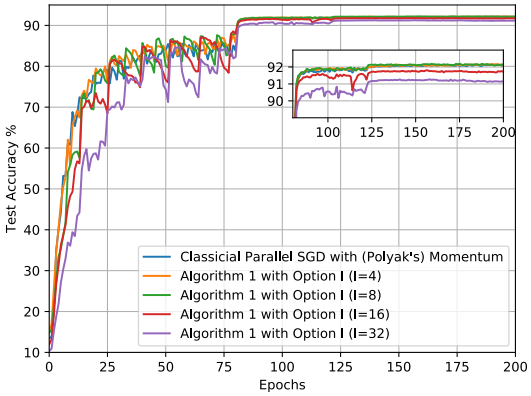
4. Experiments

In this section, we validate our theory with experiments on training ResNet (He et al., 2016) for the image classification tasks over CIFAR-10 (Krizhevsky & Hinton, 2009). Our experiments are conducted on a machine with 8 NVIDIA P100 GPUs. The local batch size at each GPU is 64. The learning rate is initialized to 0.3 and is divided by 10 when all GPUs jointly access 80 and 120 epochs.[5] The momentum coefficient , i.e., β in Algorithm 1, is set to 0.9 for both Polyak's and Nesterov's momentum. All algorithms are implemented using PyTorch 0.4. To study how communication skipping affects the convergence of Algorithm 1, we run Algorithm 1 with $I \in \{4, 8, 16, 32\}$ and compare the test accuracy convergence between Algorithm 1 and the classical parallel mini-batch SGD with momentum, which uses an inter-node communication step at every iteration and can be interpreted as Algorithm 1 with $I = 1$. Figure 2 plots the convergence of training loss and test accuracy in terms of the number

[5] Such decaying learning rates are used to achieve good test accuracy by practitioners (He et al., 2016). This deviates from the constant learning rates used in the theory to establish the linear speedup. On one hand, it is possible follow the analysis techniques in (Yu et al., 2018) to establish a similar linear speedup for decaying learning rates. On the other hand, our supplement 6.7.1 reports extra experiments to verify the linear speedup of Algorithm 1 using constant learning rates faithful to the current theory.

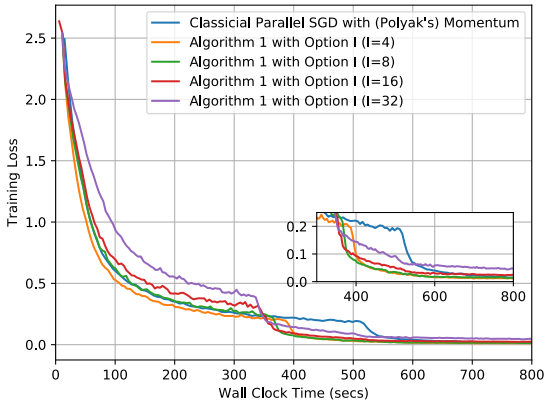


(a) Training loss v.s. epochs.

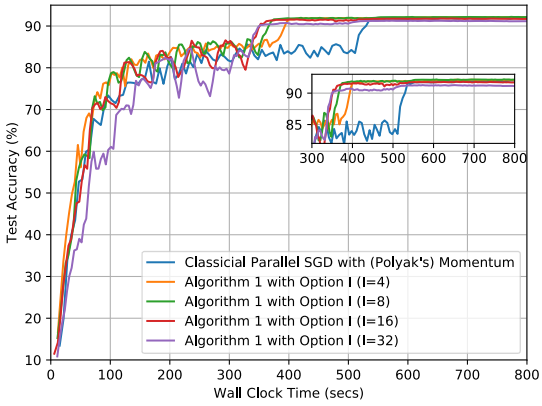


(b) Test accuracy v.s. epochs.

Figure 2: Algorithm 1 with Option I: convergence v.s. epochs for ResNet56 over CIFAR10.



(a) Training loss v.s. wall clock time.



(b) Test accuracy v.s. wall clock time.

Figure 3: Algorithm 1 with Option I: convergence v.s. wall clock time for ResNet56 over CIFAR10.

of epochs that are jointly accessed by all used GPUs. That is, if the x-axis value is 8, then each GPU access 1 epoch of training data. The same convention is followed by other figures for multiple GPU training in this paper. By plotting the convergence in terms of the number of epochs that are jointly accessed by all GPUs, we can verify the $O(\frac{1}{\sqrt{NT}})$ convergence for Algorithm 1 proven in this paper. To verify the benefit of skipping communication in our Algorithm 1, Figure 3 plots the convergence of training loss and test accuracy in terms of the wall clock time. Since Algorithm 1 with $I > 0$ skip communication steps, it is much faster than the classical parallel momentum SGD when measured by the wall clock time.

More numerical experiments on training ResNet over ImageNet (Deng et al., 2009), comparisons with a model averaging strategy suggested in (Seide & Agarwal, 2016; Wang & Joshi, 2018b), and Algorithm 1 with Nesterov's momentum are available in Supplement 6.7.

5. Conclusion

This paper considers parallel restarted SGD with momentum and prove that it can achieve $O(1/\sqrt{NT})$ convergence with $O(N^{3/2}T^{1/2})$ or $O(N^{3/4}T^{3/4})$ communication rounds depending whether each node accesses identical objective functions $f_i(\mathbf{x})$ or not. We further show that distributed momentum SGD with decentralized communication can achieve $O(1/\sqrt{NT})$ convergence.

References

Aji, A. F. and Heafield, K. Sparse communication for distributed gradient descent. In *Conference on Empir-*

ical Methods in Natural Language Processing (EMNLP), 2017.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Assran, M., Loizou, N., Ballas, N., and Rabbat, M. Stochastic gradient push for distributed deep learning. *arXiv:1811.10792*, 2018.

Chen, K. and Huo, Q. Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(165–202), 2012.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2009.

Ghadimi, E., Feyzmahdavian, H. R., and Johansson, M. Global convergence of the heavy-ball method for convex optimization. *arXiv:1412.7457*, 2014.

Ghadimi, S. and Lan, G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

Horn, R. A. and Johnson, C. R. *Matrix Analysis*. Cambridge University Press, 1985.

Jiang, P. and Agrawal, G. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Jiang, Z., Balu, A., Hegde, C., and Sarkar, S. Collaborative deep learning in fixed topology networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Lian, X., Huang, Y., Li, Y., and Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Lin, T., Stich, S. U., and Jaggi, M. Don't use large mini-batches, use local SGD. *arXiv:1808.07217*, 2018.

Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Science & Business Media, 2004.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Seide, F. and Agarwal, A. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Annual*

Conference of the International Speech Communication Association (INTERSPEECH), 2014.

Stich, S. U. Local SGD converges fast and communicates little. *arXiv:1805.09767*, 2018.

Strom, N. Scalable distributed DNN training using commodity GPU cloud computing. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2015.

Su, H., Chen, H., and Xu, H. Experiments on parallel training of deep neural network using model averaging. *arXiv:1507.01239v3*, 2018.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning (ICML)*, 2013.

Tang, H., Lian, X., Yan, M., Zhang, C., and Liu, J. D^2: Decentralized training over decentralized data. *arXiv:1803.07068*, 2018.

Wang, J. and Joshi, G. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *arXiv:1808.07576*, 2018a.

Wang, J. and Joshi, G. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. *arXiv:1810.08313*, 2018b.

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Yan, Y., Yang, T., Li, Z., Lin, Q., and Yang, Y. A unified analysis of stochastic momentum methods for deep learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Yu, H., Yang, S., and Zhu, S. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. *arXiv:1807.06629*, 2018.

Zavriev, S. and Kostyuk, F. Heavy-ball method in nonconvex optimization problems. *Computational Mathematics and Modeling*, 4(4):336–341, 1993.

Zhou, F. and Cong, G. On the convergence properties of a K-step averaging stochastic gradient descent algorithm for nonconvex optimization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.