Asynchronous Decentralized Parallel Stochastic Gradient Descent

Xiangru Lian [1 *] **Wei Zhang** [2 *] **Ce Zhang** [3] **Ji Liu** [4 1]

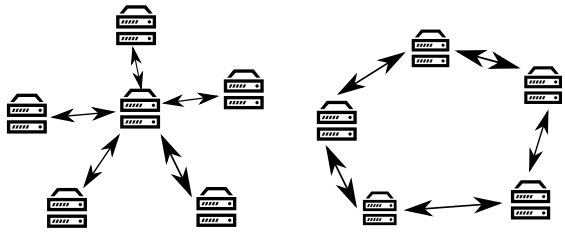
Abstract

Most commonly used distributed machine learning systems are either synchronous or centralized asynchronous. Synchronous algorithms like AllReduce-SGD perform poorly in a heterogeneous environment, while asynchronous algorithms using a parameter server suffer from 1) communication bottleneck at parameter servers when workers are many, and 2) significantly worse convergence when the traffic to parameter server is congested. *Can we design an algorithm that is robust in a heterogeneous environment, while being communication efficient and maintaining the best-possible convergence rate?* In this paper, we propose an asynchronous decentralized stochastic gradient decent algorithm (AD-PSGD) satisfying all above expectations. Our theoretical analysis shows AD-PSGD converges at the optimal $O(1/\sqrt{K})$ rate as SGD and has linear speedup w.r.t. number of workers. Empirically, AD-PSGD outperforms the best of decentralized parallel SGD (D-PSGD), asynchronous parallel SGD (A-PSGD), and standard data parallel SGD (AllReduce-SGD), often by orders of magnitude in a heterogeneous environment. When training ResNet-50 on ImageNet with up to 128 GPUs, AD-PSGD converges (w.r.t epochs) similarly to the AllReduce-SGD, but each epoch can be up to 4-8× faster than its synchronous counterparts in a network-sharing HPC environment.

1 Introduction

It often takes hours to train large deep learning tasks such as ImageNet, even with hundreds of GPUs (Goyal et al., 2017). At this scale, how workers communicate becomes a crucial design choice. Most existing systems such as TensorFlow (Abadi et al., 2016), MXNet (Chen et al., 2015), and CNTK (Seide and Agarwal, 2016) support two communication modes: (1) synchronous communication via parameter servers or AllReduce, or (2) asynchronous communication via parameter servers. When there are stragglers (i.e., slower workers) in the system, which is common especially at the scale of hundreds devices, asynchronous approaches are more robust. However, most asynchronous implementations have a *centralized* design, as illustrated in Figure 1(a) — a central server holds the shared model for all other workers. Each worker calculates its own gradients and updates the shared model asynchronously. The parameter server may become a communication bottleneck and slow down the convergence. We focus on the question: *Can we remove the central server bottleneck in asynchronous distributed learning systems while maintaining the best possible convergence rate?*

Recent work (Lian et al., 2017) shows that *synchronous de-*



(a) Centralized network. (b) Decentralized (ring) network.

Figure 1. Centralized network and decentralized network.

	Communication complexity[a] (n.t./n.h.)	Idle time
S-PSGD (Ghadimi et al., 2016)	Long ($O(n)$/$O(n)$)	Long
A-PSGD (Lian et al., 2015)	Long ($O(n)$/$O(n)$)	Short
AllReduce-SGD (Luehr, 2016)	Medium ($O(1)$/$O(n)$)	Long
D-PSGD (Lian et al., 2017)	Short ($O(\deg(G))$/$O(\deg(G))$)	Long
AD-PSGD (this paper)	Short ($O(\deg(G))$/$O(\deg(G))$)	Short

[a] n.t. means number of gradients/models transferred at the busiest worker per n (minibatches of) stochastic gradients updated. n.h. means number of handshakes at the busiest worker per n (minibatches of) stochastic gradients updated.

Table 1. Comparison of different distributed machine learning algorithms on a network graph G. *Long idle time* means in each iteration the whole system needs to wait for the slowest worker. *Short idle time* means the corresponding algorithm breaks this synchronization per iteration. Note that if G is a ring network as required in AllReduce-SGD, $O(\deg(G)) = O(1)$.

[*]Equal contribution [1]Department of Computer Science, University of Rochester [2]IBM T. J. Watson Research Center [3]Department of Computer Science, ETH Zurich [4]Tencent AI lab, Seattle, USA. Correspondence to: Xiangru Lian <admin@mail.xrlian.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

centralized parallel stochastic gradient descent (D-PSGD) can achieve comparable convergence rate as its centralized counterparts without any central bottleneck. Figure 1-(b) illustrates one communication topology of D-PSGD in which each worker only talks to its neighbors. However, the synchronous nature of D-PSGD makes it vulnerable to stragglers because of the synchronization barrier at each iteration among *all* workers. *Is it possible to get the best of both worlds of asynchronous SGD and decentralized SGD?*

In this paper, we propose the *asynchronous decentralized* parallel stochastic gradient decent algorithm (AD-PSGD) that is theoretically justified to keep the advantages of both asynchronous SGD and decentralized SGD. In AD-PSGD, workers do not wait for all others and only communicate in a decentralized fashion. AD-PSGD can achieve linear speedup with respect to the number of workers and admit a convergence rate of $O(1/\sqrt{K})$, where K is the number of updates. This rate is consistent with D-PSGD and centralized parallel SGD. By design, AD-PSGD enables wait-free computation and communication, which ensures *AD-PSGD always converges better (w.r.t epochs or wall time) than D-PSGD as the former allows much more frequent information exchanging.*

In practice, we found that AD-PSGD is particularly useful in heterogeneous computing environments such as cloud-computing, where computing/communication devices' speed often varies. We implement AD-PSGD in Torch and MPI and evaluate it on an IBM S822LC cluster of up to 128 P100 GPUs. We show that, on real-world datasets such as ImageNet, AD-PSGD has the same empirical convergence rate as its centralized and/or synchronous counterpart. In heterogeneous environments, AD-PSGD can be faster than its fastest synchronous counterparts by orders of magnitude. On an HPC cluster with homogeneous computing devices but shared network, AD-PSGD can still outperform its synchronous counterparts by 4X-8X.

Both the theoretical analysis and system implementations of AD-PSGD are non-trivial, and they form the two technical contributions of this work.

2 Related work

We review related work in this section. In the following, K and n refer to the number of iterations and the number of workers, respectively. A comparison of the algorithms can be found in Table 1.

The *Stochastic Gradient Descent* (**SGD**) Nemirovski et al. (2009); Moulines and Bach (2011); Ghadimi and Lan (2013) is a powerful approach to solve large scale machine learning problems, with the optimal convergence rate $O(1/\sqrt{K})$ on nonconvex problems.

For *Synchronous Parallel Stochastic Gradient Descent* (**S-PSGD**), every worker fetches the model saved in a parameter server and computes a minibatch of stochastic gradients. Then they push the stochastic gradients to the parameter server. The parameter server synchronizes all the stochastic gradients and update their average into the model saved in the parameter server, which completes one iteration. The convergence rate is proved to be $O(1/\sqrt{nK})$ on nonconvex problems (Ghadimi et al., 2016). Results on convex objectives can be found in Dekel et al. (2012).

The *Asynchronous Parallel Stochastic Gradient Descent* (**A-PSGD**) (Recht et al., 2011; Agarwal and Duchi, 2011; Feyzmahdavian et al., 2016; Paine et al., 2013) breaks the synchronization in S-PSGD by allowing workers to use stale weights to compute gradients. On nonconvex problems, when the staleness of the weights used is upper bounded, A-PSGD is proved to admit the same convergence rate as S-PSGD (Lian et al., 2015; 2016).

In *AllReduce Stochastic Gradient Descent implementation* (**AllReduce-SGD**) (Luehr, 2016; Patarasuk and Yuan, 2009; MPI contributors, 2015), the update rule per iteration is exactly the same as in S-PSGD, so they share the same convergence rate. However, there is no parameter server and all the workers use AllReduce to synchronize the stochastic gradients. In this procedure, only $O(1)$ amount of gradient is sent/received per worker, but $O(n)$ handshakes are needed on each worker. This makes AllReduce slow on high latency network. Since we still have synchronization in each iteration, the idle time is still high as in S-PSGD.

In *Decentralized Parallel Stochastic Gradient Descent* (**D-PSGD**) (Lian et al., 2017), all workers are connected with a network that forms a connected graph G. Every worker has its local copy of the model. In each iteration, all workers compute stochastic gradients locally and at the same time average its local model with its neighbors. Finally the locally computed stochastic gradients are updated into the local models. In this procedure, the busiest worker only sends/receives $O(\deg(G))$ models and has $O(\deg(G))$ handshakes per iteration. The idle time is still high in D-PSGD because all workers need to finish updating before stepping into the next iteration. Before Lian et al. (2017) there are also previous studies on decentralized stochastic algorithms (both synchronous and asynchronous versions) though *none of them is proved to have speedup when the number of workers increases*. For example, Lan et al. (2017) proposed a decentralized stochastic primal-dual type algorithm with a computational complexity of $O(n/\epsilon^2)$ for general convex objectives and $O(n/\epsilon)$ for strongly convex objectives. Sirb and Ye (2016) proposed an asynchronous decentralized stochastic algorithm with a $O(n/\epsilon^2)$ complexity for convex objectives. These bounds do not imply any speedup for decentralized algo-

rithms. Bianchi et al. (2013) proposed a similar decentralized stochastic algorithm. The authors provided a convergence rate for the consensus of the local models when the local models are bounded. However, they did not provide the convergence rate to the solution. A very recent paper (Tang et al., 2018) extended D-PSGD so that it works better on data with high variance. Ram et al. (2010) proposed an asynchronous subgradient variations of the decentralized stochastic optimization algorithm for convex problems. The asynchrony was modeled by viewing the update event as a Poisson process and the convergence to the solution was shown. Srivastava and Nedic (2011); Sundhar Ram et al. (2010) are similar. The main differences from this work are 1) we take the situation where a worker calculates gradients based on old model into consideration, which is the case in the asynchronous setting; 2) we prove that our algorithm can achieve linear speedup when we increase the number of workers, which is important if we want to use the algorithm to accelerate training; 3) Our implementation guarantees deadlock-free, wait-free computation and communication. Nair and Gupta (2017) proposed another distributed stochastic algorithm, but it requires a centralized arbitrator to decide which two workers are exchanging weights and it lacks convergence analysis.

We next briefly review *decentralized algorithms*. Decentralized algorithms were initially studied by the control community for solving the consensus problem where the goal is to compute the mean of all the data distributed on multiple nodes (Boyd et al., 2005; Carli et al., 2010; Aysal et al., 2009; Fagnani and Zampieri, 2008; Olfati-Saber et al., 2007; Schenato and Gamba, 2007). For decentralized algorithms used for optimization problems, Lu et al. (2010) proposed two non-gradient-based algorithms for solving one-dimensional unconstrained convex optimization problems. (Mokhtari and Ribeiro, 2016) proposed a fast decentralized variance reduced algorithm for strongly convex optimization problems. (Yuan et al., 2016) studied decentralized gradient descent on convex and strongly convex objectives. The subgradient version was considered in Nedic and Ozdaglar (2009); Ram et al. (2009). The algorithm is intuitive and easy to understand. However, the limitation of the algorithm is that it does not converge to the exact solution because the exact solution is not a fixed point of the algorithm's update rule. This issue was fixed later by Shi et al. (2015a); Wu et al. (2016) by using the gradients of last two instead of one iterates in each iteration, which was later improved in Shi et al. (2015b); Li et al. (2017) by considering proximal gradients. Decentralized ADMM algorithms were analyzed in Zhang and Kwok (2014); Shi et al.; Aybat et al. (2015). Wang et al. (2016) develops a decentralized algorithm for recursive least-squares problems.

3 Algorithm

We introduce the AD-PSGD algorithm in this section.

Definitions and notations Throughout this paper, we use the following notation and definitions:

- $\|\cdot\|$ denotes the vector ℓ_2 norm or the matrix spectral norm depending on the argument.
- $\|\cdot\|_F$ denotes the matrix Frobenius norm.
- $\nabla f(\cdot)$ denotes the gradient of a function f.
- $\mathbf{1}_n$ denotes the column vector in $\mathbb{R}^n$ with 1 for all elements.
- f^* denotes the optimal solution to (1).
- $\lambda_i(\cdot)$ denotes the i-th largest eigenvalue of a matrix.
- e_i denotes the ith element of the standard basis of $\mathbb{R}^n$.

3.1 Problem definition

The decentralized communication topology is represented as an undirected graph: (V, E), where $V := \{1, 2, \ldots, n\}$ denotes the set of n workers and $E \subseteq V \times V$ is the set of the edges in the graph. Each worker represents a machine/gpu owning its local data (or a sensor collecting local data online) such that each worker is associated with a local loss function

$$f_i(x) := \mathbb{E}_{\xi \sim \mathcal{D}_i} F_i(x; \xi),$$

where $\mathcal{D}_i$ is a distribution associated with the local data at worker i and ξ is a data point sampled via $\mathcal{D}_i$. The edge means that the connected two workers can exchange information. For the AD-PSGD algorithm, the overall optimization problem it solves is

$$\min_{x \in \mathbb{R}^N} f(x) := \mathbb{E}_{i \sim \mathcal{I}} f_i(x) = \sum_{i=1}^{n} p_i f_i(x), \qquad (1)$$

where p_i's define a distribution, that is, $p_i \geq 0$ and $\sum_i p_i = 1$, and p_i indicates the updating frequency of worker i or the percentage of the updates performed by worker i. The faster a worker, the higher the corresponding p_i. The intuition is that if a worker is faster than another worker, then the faster worker will run more epochs given the same amount of time, and consequently the corresponding worker has a larger impact.

Remark 1. *To solve the common form of objectives in machine learning using AD-PSGD*

$$\min_{x \in \mathbb{R}^N} \mathbb{E}_{\xi \sim \mathcal{D}} F(x; \xi),$$

we can appropriately distribute data such that Eq. (1) solves the target objective above:

Strategy-1 *Let $\mathcal{D}_i = \mathcal{D}$ and $\mathcal{D}$, that is, all worker can access all data, and consequently $F_i(\cdot;\cdot) = F(\cdot;\cdot)$, that is, all $f_i(\cdot)$'s are the same;*

Strategy-2 *Split the data into all workers appropriately such that the portion of data is p_i on worker i and define $\mathcal{D}_i$ to be the uniform distribution over the assigned data samples.*

3.2 AD-PSGD algorithm

The AD-PSGD algorithm can be described in the following: each worker maintains a local model x in its local memory

and (using worker i as an example) repeats the following steps:

- **Sample data:** Sample a mini-batch of training data denoted by $\{\xi_m^i\}_{m=1}^M$, where M is the batch size.
- **Compute gradients:** Use the sampled data to compute the stochastic gradient $\sum_{m=1}^M \nabla F(\hat{x}^i; \xi_m^i)$, where $\hat{x}^i$ is read from the model in the local memory.
- **Gradient update:** Update the model in the local memory by $x^i \leftarrow x^i - \gamma \sum_{m=1}^M \nabla F(\hat{x}^i; \xi_m^i)$. Note that $\hat{x}^i$ may not be the same as x^i as it may be modified by other workers in the **averaging** step.
- **Averaging:** Randomly select a neighbor (e.g. worker i') and average the local model with the worker i''s model $x^{i'}$ (both models on both workers are updated to the averaged model). More specifically, $x^i, x^{i'} \leftarrow \frac{x^i}{2} + \frac{x^{i'}}{2}$.

Note that each worker runs the procedure above on its own without any global synchronization. This reduces the idle time of each worker and the training process will still be fast even if part of the network or workers slow down.

The **averaging** step can be generalized into the following update for all workers:

$$[x^1, x^2, \ldots, x^n] \leftarrow [x^1, x^2, \ldots, x^n]W$$

where W can be an arbitrary doubly stochastic matrix. This generalization gives plenty flexibility to us in implementation without hurting our analysis.

All workers run the procedure above simultaneously, as shown in Algorithm 1. We use a virtual counter k to denote the iteration counter – every single **gradient update** happens no matter on which worker will increase k by 1. i_k denotes the worker performing the kth update.

3.3 Implementation details

We briefly describe two interesting aspects of system designs and leave more discussions to Appendix A.

3.3.1 Deadlock avoidance

A naive implementation of the above algorithm may cause deadlock — the averaging step needs to be atomic and involves updating two workers (the selected worker and one of its neighbors). As an example, given three fully connected workers A, B, and C, A sends its local model x_A to B and waits for x_B from B; B has already sent out x_B to C and waits for C's response; and C has sent out x_C to A and waits for x_A from A.

We prevent the deadlock in the following way: The communication network is designed to be a bipartite graph, that is, the worker set V can be split into two disjoint sets A (active set) and P (passive set) such that any edge in the graph connects one worker in A and one worker in P. Due to the property of the bipartite graph, the neighbors of any active worker can only be passive workers and the neighbors of any passive worker can only be active workers. This implementation avoids deadlock but still fits in the general algorithm Algorithm 1 we are analyzing. We leave more discussions and a detailed implementation for wait-free training to Appendix A.

3.3.2 Communication topology

The simplest realization of AD-PSGD algroithm is a ring-based topology. To accelerate information exchanging, we also implement a communication topology in which each sender communicates with a reciever that is $2^i + 1$ hops away in the ring, where i is an integer from 0 to $\log(n-1)$ (n is the number of learners). It is easy to see it takes at most $O(\log(n))$ steps for any pair of workers to exchange information instead of $O(n)$ in the simple ring-based topology. In this way, ρ (as defined in Section 4) becomes smaller and the scalability of AD-PSGD improves. This implementation also enables robustness against slow or failed network links because there are multiple routes for a worker to disseminate its information.

Algorithm 1 AD-PSGD (logical view)

Require: Initialize local models $\{x_0^i\}_{i=1}^n$ with the same initialization, learning rate γ, batch size M, and total number of iterations K.

1: **for** $k = 0, 1, \ldots, K-1$ **do**
2: Randomly sample a worker i_k of the graph G and randomly sample an averaging matrix W_k which can be dependent on i_k.
3: Randomly sample a batch
$$\xi_k^{i_k} := (\xi_{k,1}^{i_k}, \xi_{k,2}^{i_k}, \ldots, \xi_{k,M}^{i_k})$$
from local data of the i_k-th worker.
4: Compute the stochastic gradient locally
$$g_k(\hat{x}_k^{i_k}; \xi_k^{i_k}) := \sum_{j=1}^M \nabla F(\hat{x}_k^{i_k}; \xi_{k,j}^{i_k})$$
.
5: Average local models by [a]
$$[x_{k+1/2}^1, x_{k+1/2}^2, \ldots, x_{k+1/2}^n] \leftarrow [x_k^1, x_k^2, \ldots, x_k^n]W_k$$
6: Update the local model
$$x_{k+1}^{i_k} \leftarrow x_{k+1/2}^{i_k} - \gamma g_k(\hat{x}_k^{i_k}; \xi_k^{i_k}),$$
$$x_{k+1}^j \leftarrow x_{k+1/2}^j, \forall j \neq i_k.$$
7: **end for**
8: Output the average of the models on all workers for inference.

[a] Note that Line 4 and Line 5 can run in parallel.

4 Theoretical analysis

In this section we provide theoretical analysis for the AD-PSGD algorithm. We will show that the convergence rate of AD-PSGD is consistent with SGD and D-PSGD.

Note that by counting each update of stochastic gradients as one iteration, the update of each iteration in Algorithm 1 can be viewed as

$$X_{k+1} = X_k W_k - \gamma \partial g(\hat{X}_k; \xi_k^{i_k}, i_k),$$

where k is the iteration number, x_k^i is the local model of the

ith worker at the kth iteration, and

$$X_k = \left[\begin{array}{ccc} x_k^1 & \cdots & x_k^n \end{array}\right] \in \mathbb{R}^{N\times n},$$
$$\hat{X}_k = \left[\begin{array}{ccc} \hat{x}_k^1 & \cdots & \hat{x}_k^n \end{array}\right] \in \mathbb{R}^{N\times n},$$
$$\partial g(\hat{X}_k; \xi_k^{i_k}, i_k) = \left[0 \ \cdots \ 0 \ \sum_{j=1}^M \nabla F(\hat{x}_k^{i_k}, \xi_{k,j}^{i_k}) \ 0 \ \cdots \ 0\right] \in \mathbb{R}^{N\times n},$$

and $\hat{X}_k = X_{k-\tau_k}$ for some nonnegative integer τ_k.

Assumption 1. *Throughout this paper, we make the following commonly used assumptions:*

1. ***Lipschitzian gradient:*** *All functions $f_i(\cdot)$'s are with L-Lipschitzian gradients.*
2. ***Doubly stochastic averaging:*** *W_k is doubly stochastic for all k.*
3. ***Spectral gap:*** *There exists a $\rho \in [0,1)$ such that*[1]
$$\max\{|\lambda_2(\mathbb{E}[W_k^\top W_k])|, |\lambda_n(\mathbb{E}[W_k^\top W_k])|\} \leq \rho, \forall k. \quad (2)$$
4. ***Unbiased estimation:*** [2]
$$\mathbb{E}_{\xi\sim\mathcal{D}_i} \nabla F(x;\xi) = \nabla f_i(x), \quad (3)$$
$$\mathbb{E}_{i\sim\mathcal{I}}\mathbb{E}_{\xi\sim\mathcal{D}_i} \nabla F(x;\xi) = \nabla f(x). \quad (4)$$
5. ***Bounded variance:*** *Assume the variance of the stochastic gradient*
$$\mathbb{E}_{i\sim\mathcal{I}}\mathbb{E}_{\xi\sim\mathcal{D}_i}\|\nabla F(x;\xi) - \nabla f(x)\|^2$$
is bounded for any x with i sampled from the distribution $\mathcal{I}$ and ξ from the distribution $\mathcal{D}_i$. This implies there exist constants σ and ς such that
$$\mathbb{E}_{\xi\sim\mathcal{D}_i}\|\nabla F(x,\xi) - \nabla f_i(x)\|^2 \leqslant \sigma^2, \forall i, \forall x. \quad (5)$$
$$\mathbb{E}_{i\sim\mathcal{I}}\|\nabla f_i(x) - \nabla f(x)\|^2 \leqslant \varsigma^2, \forall x. \quad (6)$$
Note that if all workers can access all data, then $\varsigma = 0$.
6. ***Dependence of random variables:*** *$\xi_k, i_k, k \in \{0,1,2,\ldots\}$ are independent random variables. W_k is a random variable dependent on i_k.*
7. ***Bounded staleness:*** *$\hat{X}_k = X_{k-\tau_k}$ and there exists a constant T such that $\max_k \tau_k \leqslant T$.*

Throughout this paper, we define the following notations for simpler notation

$$\bar{\rho} := \frac{n-1}{n}\left(\frac{1}{1-\rho} + \frac{2\sqrt{\rho}}{(1-\sqrt{\rho})^2}\right),$$
$$C_1 := 1 - 24M^2L^2\gamma^2\left(T\frac{n-1}{n} + \bar{\rho}\right),$$
$$C_2 := \frac{\gamma M}{2n} - \frac{\gamma^2 LM^2}{n^2} - \frac{2M^3L^2T^2\gamma^3}{n^3} - \frac{4\left(\frac{6\gamma^2L^3M^2}{n^2} + \frac{\gamma M}{n}L^2 + \frac{12M^3L^4T^2\gamma^3}{n^3}\right)M^2\gamma^2(T\frac{n-1}{n} + \bar{\rho})}{C_1},$$
$$C_3 := \frac{1}{2} + \frac{2\left(6\gamma^2L^2M^2 + \gamma nML + \frac{12M^3L^3T^2\gamma^3}{n}\right)\bar{\rho}}{C_1} + \frac{LT^2\gamma M}{n}.$$

[1] A smaller ρ means a faster information spreading in the network, leading to a faster convergence.

[2] Note that this is easily satisfied when all workers can access all data so that $\mathbb{E}_{\xi\sim\mathcal{D}_i}\nabla F(x;\xi) = \nabla f(x)$. When each worker can only access part of the data, we can also meet these assumptions by appropriately distributing data.

Under Assumption 1 we have the following results:

Theorem 1 (Main theorem)**.** *While $C_3 \leqslant 1$ and $C_2 \geqslant 0$ and $C_1 > 0$ are satisfied we have*

$$\frac{\sum_{k=0}^{K-1}\mathbb{E}\left\|\nabla f\left(\frac{X_k\mathbf{1}_n}{n}\right)\right\|^2}{K} \leqslant \frac{2(\mathbb{E}f(x_0) - \mathbb{E}f^*)n}{\gamma KM} + \frac{2\gamma L(\sigma^2 + 6M\varsigma^2)}{n}.$$

Noting that $\frac{X_n\mathbf{1}_n}{n} = \frac{1}{n}\sum_{i=1}^n x_k^i$, this theorem characterizes the convergence of the average of all local models. By appropriately choosing the learning rate, we obtain the following corollary

Corollary 2. *Let $\gamma = \frac{n}{10ML + \sqrt{\sigma^2 + 6M\varsigma^2}\sqrt{KM}}$. We have the following convergence rate*

$$\frac{\sum_{k=0}^{K-1}\mathbb{E}\left\|\nabla f\left(\frac{X_k\mathbf{1}_n}{n}\right)\right\|^2}{K} \leqslant \frac{20(f(x_0) - f^*)L}{K} + \frac{2(f(x_0) - f^* + L)\sqrt{\sigma^2/M + 6\varsigma^2}}{\sqrt{K}} \quad (7)$$

if the total number of iterations is sufficiently large, in particular,

$$K \geqslant \frac{ML^2n^2}{\sigma^2 + 6M\varsigma^2}\max\left\{192\left(T\frac{n-1}{n} + \bar{\rho}\right), \frac{64T^4}{n^2}, \right. \quad (8)$$
$$\left. 1024n^2\bar{\rho}^2, \frac{\left(8\sqrt{6}T^{2/3} + 8\right)^2\left(T + \bar{\rho}\frac{n}{n-1}\right)^{2/3}(n-1)^{1/2}}{n^{1/6}}\right\}.$$

This corollary indicates that if the iteration number is big enough, AD-PSGD's convergence rate is $O(1/\sqrt{K})$. We compare the convergence rate of AD-PSGD with existing results for SGD and D-PSGD to show the tightness of the proved convergence rate. We will also show the efficiency and the linear speedup property for AD-PSGD w.r.t. batch size, number of workers, and staleness respectively. Further discussions on communication topology and intuition will be provided at the end of this section.

Remark 2 (Consistency with SGD)**.** *Note that if $T = 0$ and $n = 1$ the proposed AD-PSGD reduces to the vanilla SGD algorithm (Nemirovski et al., 2009; Moulines and Bach, 2011; Ghadimi and Lan, 2013). Since $n = 1$, we do not have the variance among workers, that is, $\varsigma = 0$, the convergence rate becomes $O(1/K + \sigma/\sqrt{KM})$ which is consistent with the convergence rate with SGD.*

Remark 3 (Linear speedup w.r.t. batch size)**.** *When K is large enough the second term on the RHS of (7) dominates the first term. Note that the second term converges at a rate $O(1/\sqrt{MK})$ if $\varsigma = 0$, which means the convergence efficiency gets boosted with a linear rate if increase the mini-batch size. This observation indicates the linear speedup w.r.t. the batch size and matches the results of mini-batch SGD.* [3]

[3] Note that when $\varsigma^2 \neq 0$, AD-PSGD does not admit this linear speedup w.r.t. batch size. It is unavoidable because increasing

Remark 4 (Linear speedup w.r.t. number of workers)**.** *Note that every single stochastic gradient update counts one iteration in our analysis and our convergence rate in Corollary 2 is consistent with SGD / mini-batch SGD. It means that the number of required stochastic gradient updates to achieve a certain precision is consistent with SGD / mini-batch SGD, as long as the total number of iterations is large enough. It further indicates the linear speedup with respect to the number of workers n (n workers will make the iteration number advance n times faster in the sense of wall-clock time, which means we will converge n times faster). To the best of our knowledge, the linear speedup property w.r.t. to the number of workers for decentralized algorithms has not been recognized until the recent analysis for D-PSGD by Lian et al. (2017). Our analysis reveals that by breaking the synchronization AD-PSGD can maintain linear speedup, reduce the idle time, and improve the robustness in heterogeneous computing environments.*

Remark 5 (Linear speedup w.r.t. the staleness)**.** *From (8) we can also see that as long as the staleness T is bounded by $O(K^{1/4})$ (if other parameters are considered to be constants), linear speedup is achievable.*

5 Experiments

We describe our experimental methodologies in Section 5.1 and we evaluate the AD-PSGD algorithm in the following sections:

- Section 5.2: Compare AD-PSGD's convergence rate (w.r.t epochs) with other algorithms.
- Section 5.3: Compare AD-PSGD's convergence rate (w.r.t runtime) and its speedup with other algorithms.
- Section 5.4: Compare AD-PSGD's robustness to other algorithms in heterogeneous computing and heterogeneous communication environments.
- Appendix B: Evaluate AD-PSGD on IBM proprietary natural language processing dataset and model.

5.1 Experiments methodology

5.1.1 Dataset, model, and software

We use CIFAR10 and ImageNet-1K as the evaluation dataset and we use Torch-7 as our deep learning framework. We use MPI to implement the communication scheme. For CIFAR10, we evaluate both VGG (Simonyan and Zisserman, 2015) and ResNet-20 (He et al., 2016) models. VGG, whose size is about 60MB, represents a communication intensive workload and ResNet-20, whose size is about 1MB, represents a computation intensive workload. For the ImageNet-1K dataset, we use the ResNet-50 model whose size is about 100MB.

the minibatch size only decreases the variance of the stochastic gradients within each worker, while ς^2 characterizes the variance of stochastic gradient among different workers, independent of the batch size.

Table 2. Testing accuracy comparison for VGG and ResNet-20 model on CIFAR10. 16 workers in total.

	AllReduce	D-PSGD	EAMSGD	AD-PSGD
VGG	87.04%	86.48%	85.75%	88.58%
ResNet-20	90.72%	90.81%	89.82%	91.49%

Additionally, we experimented on an IBM proprietary natural language processing datasets and models (Zhang et al., 2017) in Appendix B.

5.1.2 Hardware

We evaluate AD-PSGD in two different environments:

- IBM S822LC HPC cluster: Each node with 4 Nvidia P100 GPUs, 160 Power8 cores (8-way SMT) and 500GB memory on each node. 100Gbit/s Mellanox EDR infiniband network. We use 32 such nodes.
- x86-based cluster: This cluster is a cloud-like environment with 10Gbit/s ethernet connection. Each node has 4 Nvidia P100 GPUs, 56 Xeon E5-2680 cores (2-way SMT), and 1TB DRAM. We use 4 such nodes.

5.1.3 Compared algorithms

We compare the proposed AD-PSGD algorithm to AllReduce-SGD, D-PSGD (Lian et al., 2017) and a state of the art asynchronous SGD implementation EAMSGD. (Zhang et al., 2015)[4] In EAMSGD, each worker can communicate with the parameter server less frequently by increasing the "communication period" parameter su.

5.2 Convergence w.r.t. epochs

CIFAR10 Figure 2 plots training loss w.r.t. epochs for each algorithm, which is evaluated for VGG and ResNet-20 models on CIFAR10 dataset with 16 workers. Table 2 reports the test accuracy of all algorithms.

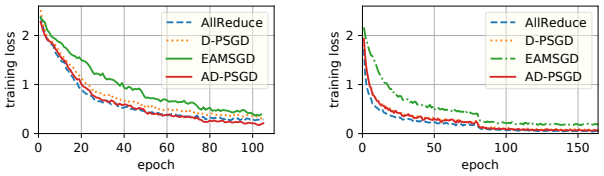
For EAMSGD, we did extensive hyper-parameter tuning to get the best possible model, where $su = 1$. We set momentum moving average to be $0.9/n$ (where n is the number of workers) as recommended in (Zhang et al., 2015) for EAMSGD.

For other algorithms, we use the following hyper-parameter setup as prescribed in (Zagoruyko, 2015) and (FAIR, 2017):

- Batch size: 128 per worker for VGG, 32 for ResNet-20.
- Learning rate: For VGG start from 1 and reduce by half every 25 epochs. For ResNet-20 start from 0.1 and decay by a factor of 10 at the 81st epoch and the 122nd epoch.
- Momentum: 0.9.
- Weight decay: 10^{-4}.

Figure 2 show that w.r.t epochs, AllReduce-SGD, D-PSGD and AD-PSGD converge similar, while ASGD converges worse. Table 2 shows AD-PSGD does not sacrifice test accuracy.

[4] In this paper, we use ASGD and EAMSGD interchangeably.



(a) VGG loss (b) ResNet-20 loss

Figure 2. Training loss comparison for VGG and ResNet-20 model on CIFAR10. AllReduce-SGD, D-PSGD and AD-PSGD converge alike, EAMSGD converges the worst. 16 workers in total.

Table 3. Testing accuracy comparison for ResNet-50 model on ImageNet dataset for AllReduce, D-PSGD, and AD-PSGD. The ResNet-50 model is trained for 90 epochs. AD-PSGD and AllReduce-SGD achieve similar model accuracy.

	AllReduce	D-PSGD	AD-PSGD
16 Workers	74.86%	74.74%	75.28%
32 Workers	74.78%	73.66%	74.66%
64 Workers	74.90%	71.18%	74.20%
128 Workers	74.78%	70.90%	74.23%

ImageNet We further evaluate the AD-PSGD's convergence rate w.r.t. epochs using ImageNet-1K and ResNet-50 model. We compare AD-PSGD with AllReduce-SGD and D-PSGD as they tend to converge better than A-PSGD.

Figure 4 and Table 3 demonstrate that w.r.t. epochs AD-PSGD converges similarly to AllReduce and converges better than D-PSGD when running with 16,32,64,128 workers. How to maintain convergence while increasing $M \times n$[5] is an active ongoing research area (Zhang et al., 2016; Goyal et al., 2017) and it is orthogonal to the topic of this paper. For 64 and 128 workers, we adopted similar learning rate tuning scheme as proposed in Goyal et al. (2017) (i.e., learning rate warm-up and linear scaling)[6] *It worths noting that we could further increase the scalability of AD-PSGD by combining learners on the same computing node as a super-learner (via Nvidia NCCL AllReduce collectives). In this way, a 128-worker system can easily scale up to 512 GPUs or more, depending on the GPU count on a node.*

Above results show AD-PSGD converges similarly to AllReduce-SGD w.r.t epochs and better than D-PSGD. Techniques used for tuning learning rate for AllReduce-SGD can be applied to AD-PSGD when batch size is large.

5.3 Speedup and convergence w.r.t runtime

On CIFAR10, Figure 3 shows the runtime convergence results on both IBM HPC and x86 system. The EAMSGD implementation deploys parameter server sharding to mitigate the network bottleneck at the parameter servers. However, the central parameter server quickly becomes a bottleneck on a slow network with a large model as shown in Figure 3-(b).

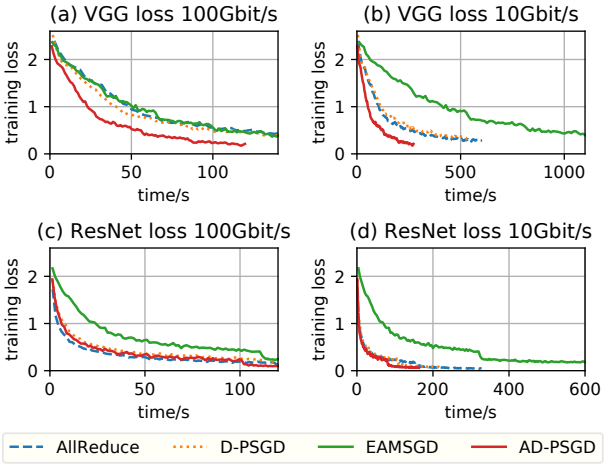


Figure 3. Runtime comparison for VGG (communication intensive) and ResNet-20 (computation intensive) models on CIFAR10. Experiments run on IBM HPC w/ 100Gbit/s network links and on x86 system w/ 10Gbit/s network links. AD-PSGD consistently converges the fastest. 16 workers in total.

Figure 5 shows the speedup for different algorithms w.r.t. number of workers. The speedup for ResNet-20 is better than VGG because ResNet-20 is a computation intensive workload.

Above results show that regardless of workload type (computation intensive or communication intensive) and communication networks (fast or slow), AD-PSGD consistently converges the fastest w.r.t. runtime and achieves the best speedup.

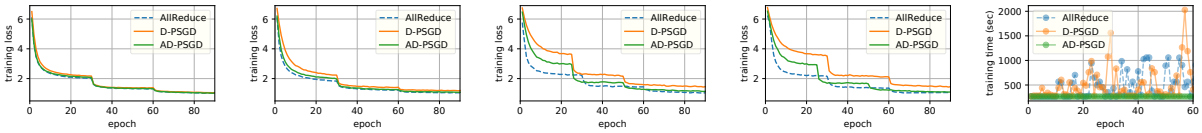
5.4 Robustness in a heterogeneous environment

In a heterogeneous environment, the speed of computation device and communication device may often vary, subject to architectural features (e.g., over/under-clocking, caching, paging), resource-sharing (e.g., cloud computing) and hardware malfunctions. Synchronous algorithms like AllReduce-SGD and D-PSGD perform poorly when workers' computation and/or communication speeds vary. Centralized asynchronous algorithms, such as A-PSGD, do poorly when the parameter server's network links slow down. In contrast, AD-PSGD localizes the impact of slower workers or network links.

On ImageNet, Figure 4e shows the epoch-wise training time of the AD-PSGD, D-PSGD and AllReduce run over 64 GPUs (16 nodes) over a reserved window of 10 hours when the job shares network links with other jobs on IBM HPC. AD-PSGD finishes each epoch in 264 seconds, whereas AllReduce-SGD and D-PSGD can take over 1000

[5] M is mini-batch size per worker and n is the number of workers

[6] In AD-PSGD, we decay the learning rate every 25 epochs instead of 30 epochs as in AllReduce.



(a) 16 workers (b) 32 workers (c) 64 workers (d) 128 workers (e) training time variation (64 workers)

Figure 4. Training loss and training time per epoch comparison for ResNet-50 model on ImageNet dataset, evaluated up to 128 workers. AD-PSGD and AllReduce-SGD converge alike, better than D-PSGD. For 64 workers AD-PSGD finishes each epoch in 264 seconds, whereas AllReduce-SGD and D-PSGD can take over 1000 sec/epoch.

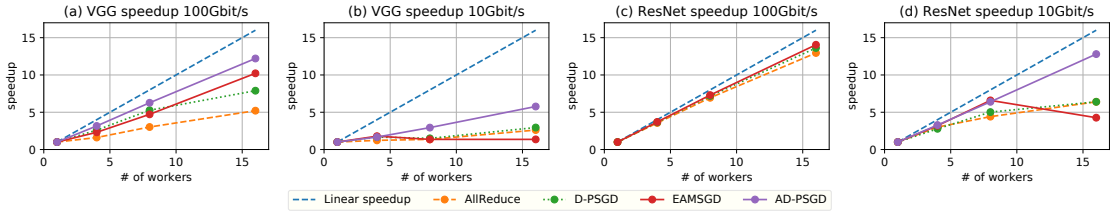
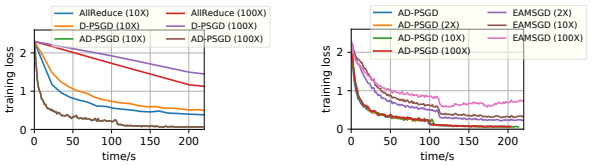


Figure 5. Speedup comparison for VGG (communication intensive) and ResNet-20 (computation intensive) models on CIFAR10. Experiments run on IBM HPC w/ 100Gbit/s network links and on x86 system w/ 10Gbit/s network links. AD-PSGD consistently achieves the best speedup.



(a) A computation device slows down by 10X-100X. (b) A network link slows down by 2X-100X.

Figure 6. Training loss for ResNet-20 model on CIFAR10, when a computation device/network link slows down by 2X-100X. kX in parentheses means a random worker's GPU or network links slow down by k-times. 16 workers in total. AD-PSGD is robust under such heterogeneous environments.

Table 4. Runtime comparison for ResNet-20 model on CIFAR10 dataset when a worker slows down by 2X-100X.

Slowdown of one node	AD-PSGD		AllReduce/D-PSGD	
	Time/epoch (sec)	Speedup	Time/epoch (sec)	Speedup
no slowdown	1.22	14.78	1.47/1.45	12.27/12.44
2X	1.28	14.09	2.6/2.36	6.93/7.64
10X	1.33	13.56	11.51/11.24	1.56/1.60
100X	1.33	13.56	100.4/100.4	0.18/0.18

sec/epoch.

We then evaluate AD-PSGD's robustness under different situations by randomly slowing down 1 of the 16 workers and its incoming/outgoing network links. Due to space limit, we will discuss the results for ResNet-20 model on CIFAR10 dataset as the VGG results are similar.

Robustness against slow computation *Figure 6a and Table 4 shows that AD-PSGD's convergence is robust against slower workers. AD-PSGD can converge faster than AllReduce-SGD and D-PSGD by orders of magnitude when there is a very slow worker.*

Robustness against slow communication Figure 6b shows that AD-PSGD is robust when one worker is connected to slower network links. In contrast, centralized asynchronous algorithm EAMSGD uses a larger communication period to overcome slower links, which significantly slows down the convergence.

These results show only AD-PSGD is robust against both heterogeneous computation and heterogeneous communication.

6 Conclusion

This paper proposes an asynchronous decentralized stochastic gradient descent algorithm (AD-PSGD). The algorithm is not only robust in heterogeneous environments by combining both decentralization and asynchronization, but it is also theoretically justified to have the same convergence rate as its synchronous and/or centralized counterparts and can achieve linear speedup w.r.t. number of workers. Extensive experiments validate the proposed algorithm.

Acknowledgment This project is supported in part by NSF CCF1718513, NEC fellowship, IBM faculty award, Swiss NSF NRP 75 407540_167266, IBM Zurich, Mercedes-Benz Research & Development North America, Oracle Labs, Swisscom, Zurich Insurance, and Chinese Scholarship Council.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *NIPS*, 2011.

N. S. Aybat, Z. Wang, T. Lin, and S. Ma. Distributed linearized alternating direction method of multipliers for composite convex consensus optimization. *arXiv preprint arXiv:1512.08122*, 2015.

T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione. Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal processing*, 2009.

P. Bianchi, G. Fort, and W. Hachem. Performance of a distributed stochastic approximation algorithm. *IEEE Transactions on Information Theory*, 2013.

S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *INFOCOM*, 2005.

R. Carli, F. Fagnani, P. Frasca, and S. Zampieri. Gossip consensus algorithms via quantized communication. *Automatica*, 2010.

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

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

F. Fagnani and S. Zampieri. Randomized consensus algorithms over large scale networks. *IEEE Journal on Selected Areas in Communications*, 2008.

FAIR. ResNet in Torch. https://github.com/facebook/fb.resnet.torch, 2017.

H. R. Feyzmahdavian, A. Aytekin, and M. Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 2016.

S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 2013.

S. Ghadimi, G. Lan, and H. Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 2016.

P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

G. Lan, S. Lee, and Y. Zhou. Communication-efficient algorithms for decentralized and stochastic optimization. *arXiv preprint arXiv:1701.03961*, 2017.

Z. Li, W. Shi, and M. Yan. A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates. *arXiv preprint arXiv:1704.07807*, 2017.

X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, 2015.

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 D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *NIPS*. Curran Associates, Inc., 2016.

X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent, 2017.

J. Lu, C. Y. Tang, P. R. Regier, and T. D. Bow. A gossip algorithm for convex consensus optimization over networks. In *ACC*. IEEE, 2010.

N. Luehr. Fast multi-gpu collectives with nccl, 2016. URL https://devblogs.nvidia.com/parallelforall/fast-multi-gpu-collectives-nccl/.

A. Mokhtari and A. Ribeiro. DSA: decentralized double stochastic averaging gradient algorithm. *Journal of Machine Learning Research*, 2016.

E. Moulines and F. R. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *NIPS*, 2011.

MPI contributors. MPI AllReduce, 2015. URL http://mpi-forum.org/docs/.

R. Nair and S. Gupta. Wildfire: Approximate synchronization of parameters in distributed deep learning. *IBM Journal of Research and Development*, 61(4/5):7:1–7:9, July 2017. ISSN 0018-8646. doi: 10.1147/JRD.2017.2709198.

A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 2009.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 2009.

R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 2007.

T. Paine, H. Jin, J. Yang, Z. Lin, and T. Huang. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*, 2013.

P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 2009.

S. S. Ram, A. Nedic, and V. V. Veeravalli. Distributed subgradient projection algorithm for convex optimization. In *ICASSP*. IEEE, 2009.

S. S. Ram, A. Nedić, and V. V. Veeravalli. Asynchronous gossip algorithm for stochastic optimization: Constant stepsize analysis. In *Recent Advances in Optimization and its Applications in Engineering*. Springer, 2010.

B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, 2011.

L. Schenato and G. Gamba. A distributed consensus protocol for clock synchronization in wireless sensor network. In *CDC*. IEEE, 2007.

F. Seide and A. Agarwal. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, 2016.

W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin. On the linear convergence of the admm in decentralized consensus optimization.

W. Shi, Q. Ling, G. Wu, and W. Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 2015a.

W. Shi, Q. Ling, G. Wu, and W. Yin. A proximal gradient algorithm for decentralized composite optimization. *IEEE Transactions on Signal Processing*, 63(22):6013–6023, 2015b.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

B. Sirb and X. Ye. Consensus optimization with delayed and stochastic gradients on decentralized networks. In *Big Data*, 2016.

K. Srivastava and A. Nedic. Distributed asynchronous constrained stochastic optimization. *IEEE Journal of Selected Topics in Signal Processing*, 2011.

S. Sundhar Ram, A. Nedić, and V. Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 2010.

H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu. D2: Decentralized training over decentralized data. *arXiv preprint arXiv:1803.07068*, 2018.

Z. Wang, Z. Yu, Q. Ling, D. Berberidis, and G. B. Giannakis. Decentralized rls with data-adaptive censoring for regressions over large-scale networks. *arXiv preprint arXiv:1612.08263*, 2016.

T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed. Decentralized consensus optimization with asynchrony and delays. *arXiv preprint arXiv:1612.00150*, 2016.

K. Yuan, Q. Ling, and W. Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 2016.

S. Zagoruyko. CIFAR VGG in Torch. https://github.com/szagoruyko/cifar.torch, 2015.

R. Zhang and J. Kwok. Asynchronous distributed admm for consensus optimization. In *ICML*, 2014.

S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.

W. Zhang, S. Gupta, and F. Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *IEEE International Conference on Data Mining*, 2016.

W. Zhang, M. Feng, Y. Zheng, Y. Ren, Y. Wang, J. Liu, P. Liu, B. Xiang, L. Zhang, B. Zhou, and F. Wang. Gadei: On scale-up training as a service for deep learning. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. The IEEE International Conference on Data Mining series(ICDM'2017), 2017.