ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240777

# Adjacent Leader Decentralized Stochastic Gradient Descent

**Haoze He**[a,1], **Jing Wang**[b,1] **and Anna Choromanska**[c,2]

[a,b,c]New York University, Tandon School of Engineering

**Abstract.** This work focuses on the decentralized deep learning optimization framework. We propose Adjacent Leader Decentralized Gradient Descent (AL-DSGD), for improving final model performance, accelerating convergence, and reducing the communication overhead of decentralized deep learning optimizers. AL-DSGD relies on two main ideas. Firstly, to increase the influence of the strongest learners on the learning system it assigns weights to different neighbor workers according to both their performance and the degree when averaging among them, and it applies a corrective force on the workers dictated by both the currently best-performing neighbor and the neighbor with the maximal degree. Secondly, to alleviate the problem of the deterioration of the convergence speed and performance of the nodes with lower degrees, AL-DSGD relies on dynamic communication graphs, which effectively allows the workers to communicate with more nodes while keeping the degrees of the nodes low. Experiments demonstrate that AL-DSGD accelerates the convergence of the decentralized state-of-the-art techniques and improves their test performance especially in the communication constrained environments. We also theoretically prove the convergence of the proposed scheme. Finally, we release to the community a highly general and concise PyTorch-based library for distributed training of deep learning models that supports easy implementation of any distributed deep learning approach ((a)synchronous, (de)centralized).

## 1 Introduction

Stochastic gradient descent (SGD) is the skeleton of most state-of-the-art (SOTA) machine learning algorithms. The stability and convergence rate of classical SGD, which runs serially at a single node, has been well studied [11, 16]. However, recently, the dramatical increase in the size of deep learning models [13, 5, 35], the amount of computations and the size of the data sets [12, 25] make it challenging to train the model on a single machine. To efficiently process these quantities of data with deep learning models, distributed SGD, which parallelizes training across multiple workers, has been successfully employed. However, centralized distributed SGD with a parameter server [9, 27, 8, 15, 10] suffers from the communication bottleneck problem when the framework either has a large number of workers or low network bandwidth [31, 30, 29, 40].

To overcome the communication bottleneck, decentralized frameworks have been proposed. Moving from centralized to decentralized learning system is a highly non-trivial task. Many popular distributed deep learning schemes, including EASGD [50], LSGD [38], as well as PyTorch embedded scheme called DataParallel [28] are only compatible with the centralized framework. In a decentralized setting, there exists no notion of the central parameter server and the communication between workers is done over the communication network with a certain topology. The decentralized approach is frequently employed and beneficial for training in various settings, including sensor networks, multi-agent systems, and federated learning on edge devices. Most of the decentralized SGD algorithms communicate over the network with pre-defined topology and utilize parameter averaging instead of gradient updates during the communication step. This is the case for the SOTA approach called decentralized parallel SGD (D-PSGD) [29], that allows each worker to send a copy of its model to its adjacent workers at every iteration, and its variants [29, 30, 43, 23]. These methods satisfy convergence guarantees in terms of iterations or communication rounds [14, 21, 33, 36, 39, 46, 49, 29, 30]. Since each worker only needs to compute an average with its neighbors, they reduce communication complexity compared to centralized methods [30, 4, 22, 29, 43]. (Neighbor workers refer to a group of workers within the distributed deep learning framework that are connected to a given worker.) Computing simple average of model parameters during the communication step effectively leads to treating all the workers over which the average is computed equally, regardless of their learning capabilities. ***What is new in this paper?*** In this paper, we propose AL-DSGD, a decentralized distributed SGD algorithm with a novel *averaging strategy* that assigns specific weights to different neighbor learners based on both their performance and degree, and applies a *corrective force* dictated by both the currently best-performing and the highest degree neighbor when training to accelerate the convergence and improve the generalization performance.

Furthermore, the convergence rate of decentralized SGD is influenced by the network topology. A dense topology demands more communication time [43], despite converging fast iteration-wise [43, 23], while a sparse topology or one with imbalanced degrees across learners (i.e., nodes have degrees that vary a lot; we will refer to this topology as imbalanced topology) results in a slower convergence in terms of iterations but incurs less communication delays [41]. Previously proposed solutions addressing the problem of accelerating convergence without sacrificing the performance of the system include bias-correction techniques [47, 20, 48, 37, 18, 48, 1], periodic global averaging or multiple partial averaging methods for reducing frequent communication [6, 2, 42, 3, 24], methods that design new topologies [34, 7, 23], or techniques utilizing the idea of

[1] Equal contribution.
[2] Corresponding author.

communicating more frequently over connectivity-critical links and at the same time using other links less frequently as is done in the SOTA algorithm called MATCHA [43, 40]. However, these proposed solutions rely on simple model averaging or gradient summation during the communication step. Moreover, many of these solutions use an optimizer that is dedicated to single GPU training and naively adapt it to distributed optimization. It still remains a challenge to design a strategy for handling the communication topology that at the same time allows for fast convergence in terms of iterations and carries low communication burden resulting in time-wise inexpensive iterations. ***What else is new in this paper?*** Our proposed AL-DSGD algorithm addresses this challenge by employing *dynamic communication graphs*. The dynamic graphs are a composite structure formed from a series of communication graphs. The communication graph and weight matrices follow established practices[41, 29]. Each matrix W(k) is symmetric and doubly stochastic, ensuring nodes converge to the same stationary point[41]. Our method switches between different communication graphs during training thus allowing workers to communicate with more neighbors than when keeping the topology fixed without increasing the total degree of the graph[3]. That equips our method with robustness to the problems faced by imbalanced and sparse topologies, and allows to achieve fast convergence and improved generalization performance in communication-constrained environments.

***What else is our contribution in this paper?*** The empirical analysis demonstrating that AL-DSGD converges faster, generalizes better, and is more stable in the communication-constrained environments compared to the SOTA approaches, and the theoretical analysis showing that the AL-DSGD algorithm that relies on dynamic communication graphs indeed achieves a sub-linear convergence rate is all new. The proposed AL-DSGD is a meta-scheme algorithm that can be applied to most decentralized SGD algorithms. Given any decentralized algorithm as a baseline, AL-DSGD can accelerate its convergence while also improve robustness to imbalanced and sparse topologies. Finally, we release a general and concise PyTorch-based library for distributed training of deep learning models that supports easy implementation of any distributed deep learning approach ((a)synchronous, (de)centralized). The library is attached to the Supplementary materials and will be released publicly.

The paper is organized as follows: Section 2 contains the preliminaries, Section 3 motivates the proposed AL-DSGD algorithm, Section 4 presents the algorithm, Section 5 captures the theoretical analysis, Section 6 reports the experimental results, and finally Section 7 concludes the paper. Supplementary materials contain proofs, additional derivations, in-depth description of the experiments, as well as additional empirical results.

## 2 Preliminaries

### 2.1 Distributed Optimization Framework

Distributed machine learning models are trained on data center clusters, where workers can communicate with each other, subject to communication time and delays. Consider a distributed SGD system with $m$ worker nodes. The model parameters are denoted by $x$ where $x \in R^d$. Each worker node $i$ sees data from its own local data distribution $D_i$. The purpose of distributed SGD is to train a model by minimizing the objective function $F(x)$ using $m$ workers. The problem can be defined as follows:

$$\min_{x\in R^d} F(x) = \min_{x\in R^d} \frac{1}{m}\sum_{i=1}^{m} F_i(x)$$
$$= \min_{x\in R^d} \frac{1}{m}\sum_{i=1}^{m} E_{s\sim D_i}[l(x;s)], \quad (1)$$

where $l(x)$ is the loss function and $F_i(x) = E_{s\sim D_i}[l(x;s)]$ is the local objective function optimized by the $i$-th worker.

[3] Total degree is the total number of links in the communication network topology.

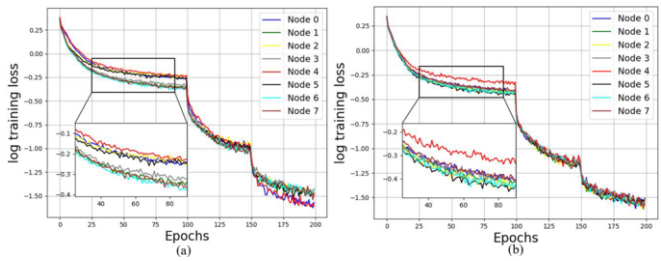


**Figure 1.** Workers with lower degree have worse performance. (a) is the performance of D-PSGD and (b) is the performance of MATCHA. Results were obtained on CIFAR-10 data set using ResNet-50.

### 2.2 Decentralized SGD (D-PSGD)

Decentralized distributed SGD can overcome the communication bottleneck problem of centralized SGD [29, 36, 43, 17]. In D-PSGD (also referred to as consensus-based distributed SGD), workers perform one local update and average their models only with neighboring workers. The update rule is given as:

$$x_{k+1,i} = \sum_{j=1}^{m} W_{ij}\left[x_{k,j} - \eta g_j(x_{k,j};\xi_{k,j})\right], \quad (2)$$

where $x_{k,j}$ denotes the model parameters of worker $j$ at iteration $k$, $\xi_{k,j}$ denotes a batch sampled from a local data distribution of worker $j$ at iteration $k$, $W \in \mathbb{R}^{m\times m}$ and $W_{ij}$ is the $(i,j)$-th element of the mixing matrix $W$ which presents the adjacency of node $i$ and $j$. $W_{ij}$ is non-zero if and only if node $i$ and node $j$ are connected. Consequently, sparse topologies would correspond to sparse mixing matrix $W$ and dense topologies correspond to matrix $W$ with most of the entries being away from 0. More details about baselines, D-PSGD and MATCHA, can be found in section §3.1.

## 3 Motivations

In this section we motivate our AL-DSGD algorithm. We start from discussing two SOTA decentralized SGD approaches, D-PSGD and MATCHA, and next show that both D-PSGD and MATCHA suffer from, what we call, the lower degree-worse performance phenomenon.

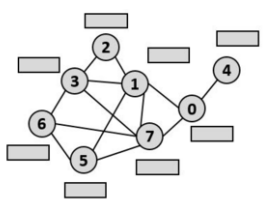


**Figure 2.** Illustration of decentralized SGD algorithm.

### 3.1 D-PSGD and MATCHA: Overview

An overview of the base communication topology for existing decentralized SGD methods can be found in Figure 2. Here, we first compare the construction of the sequence of weight matrices (or in other words mixing matrices) $\{W^{(k)}\}$ ($k$ is the iteration counter) for D-PSGD and MATCHA. In the case of the D-PSGD algorithm, the weight matrices remain the same (denoted as $W$) for the whole training process and thus the communication topology remains fixed the whole time. The communication between workers during training occurs over a computational graph, where the workers are represented as nodes and the edges denote the connections between them. The connectivity pattern is captured in $W$. Similarly to D-PSGD, MATCHA starts with a pre-defined communication network topology. In contrast to D-PSGD, MATCHA allows the system designer to set a flexible communication budget $c_b$, which represents the average frequency of communication over the links in the network. The sequence of the weight matrices $\{W^{(k)}\}$ is constructed dynamically and depends on the choice of the budget $c_b$. When $c_b$ = 1, MATCHA is equivalent to vanilla D-PSGD algorithm. When $c_b < 1$, MATCHA carefully reduces the frequency of communication over each link (how quickly this is done depends upon the importance of the link in maintaining the overall connectivity of the graph). In addition, MATCHA assigns probabilities to connections between workers, thus they may become active in some iterations.

### 3.2 Lower Degree - Worse Performance Phenomenon

| | D-PSGD | | | MATCHA | | |
|---|---|---|---|---|---|---|
| **Node** | **Loss$^1$** | **Loss$^2$** | **TEST ACC** | **Loss$^1$** | **Loss$^2$** | **TEST ACC** |
| 0 | 0.54 | **0.027**↓ | **87.95**↓ | 0.39 | 0.030 | 93.78 |
| 1 | 0.45 | 0.037 | 92.11 | 0.36 | 0.030 | 93.68 |
| 2 | 0.58 | 0.035 | 92.21 | 0.36 | 0.029 | 93.76 |
| 3 | 0.45 | 0.034 | 92.36 | 0.38 | 0.027 | 93.91 |
| 4 | **0.59** | **0.025**↓ | **87.86**↓ | **0.47** | 0.026 | 93.68 |
| 5 | 0.55 | 0.032 | 92.25 | 0.36 | 0.031 | 93.81 |
| 6 | 0.42 | 0.037 | 92.38 | 0.37 | 0.027 | 93.72 |
| 7 | 0.44 | 0.034 | 92.32 | 0.38 | 0.031 | 93.82 |

**Table 1.** Performance of D-PSGD and MATCHA on CIFAR-10. Loss$^1$ is the training loss computed at the $100^{\text{th}}$ epoch on a local data set seen by the node. Loss$^2$ is the training loss computed at the $200^{\text{th}}$ epoch on a local data set seen by the node.

Previous literature [41] explored how topology affects performance in distributed decentralized SGD optimizers. Denser networks lead to quicker error convergence in terms of iterations but introduce longer communication delays. In addition we discuss the lower degree-worse performance phenomenon in this section. In particular we show that the worker with lower degree will converge slower than the other nodes, and achieve worse final model at the end of training. We use Figure 1 and Table 1 to illustrate this phenomenon (Experiments are using topology in figure 2. Refer to Section 6 regarding experimental details). Figure 1 shows that the node denoted as Node 4 achieves the slowest convergence before the first drop of the learning rate from among all of the workers.

As seen in Figure 2, Node 4 possesses the lowest degree, with only Node 0 connected to it. Due to the weight matrix $W$ design, weaker node performance adversely affects neighbors. Notably, in D-PSGD, Node 0 and Node 4 display reduced local training loss, yet their test accuracy lags behind other nodes. This is because the training loss is calculated using a local subset seen by the worker, and a lack of communication results in the over-fitting of the local model. This experiment shows that lower degree leads to slower convergence and worse final model (this phenomenon is observed in every single experiment with different random seeds). In both D-PSGD and MATCHA, the output model is the average of models from all of the workers. Workers with lower degrees will naturally deteriorate the performance of the entire system. Therefore, a natural question arises: how to improve the performance of workers with low degrees and accelerate the overall convergence of the entire system without increasing the density of the network topology?

**Algorithm 1** Proposed AL-DSGD algorithm

1: **Initialization:**initialize local models $\{x_0^i\}_{i=1}^m$ with the **different** initialization, learning rate $\gamma$, weight matrices sequence $\{W^{(k)}\}$, and the total number of iterations $K$. Initialize communication graphs set $\{G_{(i)}\}_{i=1}^n$, each communication graph $\{G_{(i)}\}$has its own weight matrices sequence $\{W^{(k)}\}$. Pulling coefficients $\lambda_N$ and $\lambda_\tau$. Model weights coefficients $w_N$ and $w_\tau$. Split the original dataset into subsets.
2: **while** k=0, 1, 2, ... K-1 $\leq$ K **do**
3: Compute the local stochastic gradient $\nabla F_i(x_{k,i};\xi_{k,i})$ on all workers
4: For each worker, fetch neighboring models and determine the adjacent best worker $x_{k,i}^N$ and the adjacent maximum degree worker: $x_{k,i}^\tau$.
5: Update the local model with corrective force:
$$x_{k+\frac{1}{2},i} = x_{k,i} - \gamma\nabla F_i(x_{k,i};\xi_{k,i}) - \gamma\lambda_N(x_{k,i} - x_{k,i}^N) - \gamma\lambda_\tau(x_{k,i} - x_{k,i}^\tau)$$
6: Average the model with neighbors, give additional weights to worker $x_{k,i}^N$ and $x_{k,i}^\tau$:
$$x_{k+1,i} = (1 - w_N - w_\tau)\cdot(\sum_{j=1,j\neq i}^{m} W_{ij}^{(k)} x_{k,j} + W_{ii}^{(k)} x_{k+\frac{1}{2},i}) + w_N \cdot x_{k,i}^N + w_\tau \cdot x_{k,i}^\tau$$
7: Switch to new communication graph.
8: **end while**
9: **Output:**the average of all workers $\frac{1}{m}\sum_{i=1}^m x_{k,i}$

## 4 Proposed Method

We present the Adjacent Leader Decentralized Gradient Descent(AL-DSGD) in Algorithm 1 and discuss it in details below. A ***motivation example*** for Algorithm 1 can be found in figure 4. A ***visualization*** of step 6 in Algorithm 1 can be found in Figure 3 and a ***visualization*** of step 7 (dynamic communication graphs) in Algorithm 1 can be found in Figure 5. Let $k$ denote the iteration index ($k = 1, 2, \ldots, K$) and $i$ denote the index of the worker ($i = 1, 2, \ldots, m$). Let $x_{k,i}^N$ denote the best performing worker from among the workers adjacent to node $i$ at iteration $k$ and let $x_{k,i}^\tau$ denote the maximum degree worker from among the workers adjacent to node $i$ at iteration $k$. Let $\{G_{(i)}\}_{i=1}^n$ denote the dynamic communication graphs, each communication graph $\{G_{(i)}\}$has its own weight matrices sequence $\{W^{(k)}\}$. In the upcoming paragraphs of this section, we present novel contributions corresponding to Algorithm 1.

***The Corrective Force:*** To address the problem of detrimental influence of low degree nodes on the entire learning system, we first increase the influence of the workers with better performance ($x_{k,i}^N$) and larger degrees ($x_{k,i}^\tau$). Specifically, in the communication step, workers send their model parameters, local training loss, and local degree to their neighbors. At the end of the communication, we determine the adjacent best worker ($x_{k,i}^N$) based on ***training loss*** and

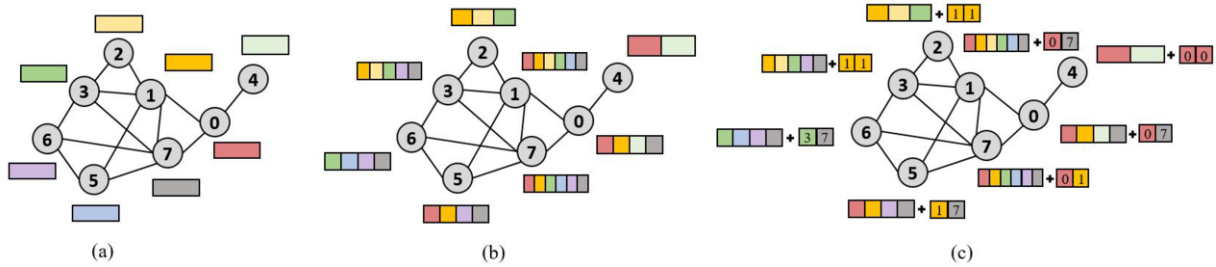


**Figure 3.** (a) The weights before communication are represented as colored blocks, where different colors correspond to different workers. (b) Previous methods simply average the training model with neighbors. Each colored block denotes the identity of workers whose parameters were taken to compute the average. (c) To illustrate AL-DSGD, we assume that the higher is the index of the worker, the worse is its performance in this iteration. For each node, in addition to averaging with neighboring models, AL-DSGD assigns additional weights to the best performing adjacent model and the maximum degree adjacent model. This is depicted as the sum, where the additional block has two pieces (the left corresponds to the best performing adjacent model and the right corresponds to the maximum degree adjacent model; the indexes of these models are also provided). For example, in the case of model 2, both the best-performing adjacent model and the maximum degree adjacent model is model 1.

the adjacent maximum degree worker ($x^{\tau}_{k,i}$) based on ***local degree*** for each node. Then, at training we introduce an additional corrective "force" pushing workers to their adjacent nodes with the largest degrees and the lowest train loss according to the following:

$$x_{k+\frac{1}{2},i} = x_{k,i} - \gamma \nabla F_i(x_{k,i}; \xi_{k,i}) - \gamma\lambda_N(x_{k,i} - x^N_{k,i}) - \gamma\lambda_\tau(x_{k,i} - x^\tau_{k,i})$$

where $\lambda_N$ and $\lambda_\tau$ are pulling coefficients, $\gamma$ is the learning rate, and $\nabla F_i(x_{k,i}; \xi_{k,i})$ is the gradient of the loss function for worker $i$ computed on parameters $x_{k,i}$ and local data sample $\xi_{k,i}$ that is seen by worker $i$ at iteration $k$. This update is done in step 5 of the Algorithm 1. Finally, note that *no additional computations or communication steps are required* to acquire the information about the best performing adjacent worker or the maximum degree adjacent worker for a given node. This is because during the training process we compute the training losses, and furthermore each worker must be aware of its degree and the degree of the adjacent worker before communication. Figure 4 intuitively illustrates that adding the corrective force can accelerate the convergence rate of the worker with low degree and worse performance.

***The Averaging Step:*** Secondly, when averaging workers, we weight them according to their degree and performance (see step 6). This is done according to the formula:

$$x_{k+1,i} = (1 - w_N - w_\tau) \cdot (W^{(k)}_{ii} x_{k+\frac{1}{2},i} + \sum_{j=1, j\neq i}^{m} W^{(k)}_{ij} x_{k,j}) + w_N \cdot x^N_{k,i} + w_\tau \cdot x^\tau_{k,i}$$

We visualize the process in Figure 3. Take node 0 as an example. Figure 3(a) to 3(b) shows the classical communication step in vanilla decentralized SGD algorithms. Since the node 7, that is adjacent to node 0, has the largest degree and node 0 itself is the best-performance worker, we increase the weight of node 7 and node 0 in Figure 3(c).

***The Dynamic Communication Graph:*** Third, instead of relying on a single communication graph, we introduce $n$ graphs with different topologies and switch between them (see Figure 5 for the special case of $n = 3$). The total degree of each graph is the same as for the baseline. When we switch the graph, we are indeed altering the physical network topology. Since distributed machine learning models are trained on data center clusters, where workers can communicate with each other, *switching communication graph won't lead to additional training time.* We randomly choose a graph to start with and switch between different graphs after each training iteration. By using the dynamic communication graph, the workers

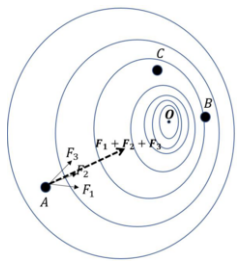


**Figure 4.** Motivating example: In Algorithm 1 step 5, Point A represents a worker model with low degree and poor performance. $F_1$ is the data batch gradient, $F_2$ is the corrective force from the best performing adjacent worker, and $F_3$ is the corrective force from the adjacent worker with the highest degree. Point B represents the best performing adjacent node to A, while Point C represents the adjacent node with the maximum degree. Point O represents the optimum. Note that $F_1 + F_2 + F_3$ directs to the optimum, highlights the benefit of corrective force in optimization.

are connected to more neighbors. This is done without increasing the total degree of the graph. This allows us to balance the expected degree of each node and avoid the poor performance of nodes with extremely low degrees.

Finally, we would like to emphasize that ALD-SGD is a meta-scheme that can be put on the top of any decentralized SGD method, including D-PSGD and MATCHA.

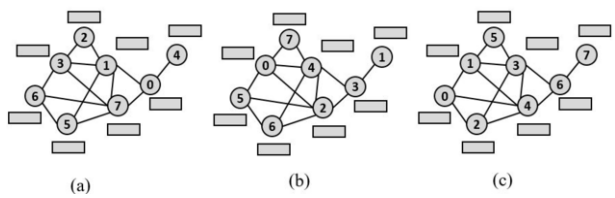


**Figure 5.** AL-DSGD with three Laplacian matrices rotates workers locations between (a), (b), and (c).

## 5 Theoretical analysis

This section offers theoretical analysis for the proposed AL-DSGD algorithm. As a meta-scheme adaptable to various decentralized SGD methods, our analysis focuses on embedding AL-DSGD with the MATCHA core (where D-PSGD is a MATCHA special case, making our analysis broadly applicable). The structure is as follows:

Section 5.1 establishes the update formula for AL-DSGD atop MATCHA, incorporating a dynamic communication graph. Notably, this subsection includes a convergence theorem. In Section 5.2, we demonstrate that, subject to specified assumptions, a range of hyperparameters $\alpha, \omega_N, \omega_\tau$, pull coefficients $\lambda_N$ and $\lambda_\tau$, along with the learning rate $\eta$, exist, resulting in AL-DSGD achieving a sub-linear convergence rate. Thus our theoretical guarantee matches that of MATCHA and D-PSGD and shows that by using dynamic graphs we do not loose in terms of convergence compared to these schemes.

### 5.1 Averaged weight matrix

From Algorithm1, the model average step for our AL-DSGD algorithm is: $\overbrace{\quad\quad\quad\quad}^{(I)}$

$$\begin{aligned} x_{k+1,i} = &(1-\omega_N-\omega_\tau)\sum_{j=1}^{m} W_{ij}^{(k)} x_{k,j} + \omega_N x_{k,i}^n + \omega_\tau x_{k,i}^\tau \\ &- \gamma(1-\omega_N-\omega_\tau) W_{ii}^{(k)} [\ \nabla F_i(x_{k,i};\xi_{k,i}) \\ &+ \lambda_N(x_{k,i}-x_{k,i}^N) + \lambda_\tau(x_{k,i}-x_{k,i}^\tau)\ ] \end{aligned} \tag{3}$$

In this section, we only consider part (I) in formula (3) without the gradient update step. We define $\widetilde{x}_{k+\frac{1}{2},i} := (I)$, and denote $X_k = [x_{k,1}, x_{k,2}, ..., x_{k,m}]$, $\widetilde{X}_{k+\frac{1}{2}} = [\widetilde{x}_{k+\frac{1}{2},1}, \widetilde{x}_{k+\frac{1}{2},2}, ..., \widetilde{x}_{k+\frac{1}{2},m}]$, $X_k^N = [x_{k,1}^N, x_{k,2}^N, ..., x_{k,m}^N]$ and $X_k^\tau = [x_{k,1}^\tau, x_{k,2}^\tau, ..., x_{k,m}^\tau]$, we have

$$\begin{aligned} \widetilde{X}_{k+\frac{1}{2}} &= X_k \widetilde{W}^{(k)} \\ \widetilde{W}^{(k)} &= (1-\omega_N-\omega_\tau)W^{(k)} + \omega_N A_k^N + \omega_\tau A_k^\tau \\ W^{(k)} &= 1 - \alpha L^{(k)}. \end{aligned} \tag{4}$$

where $L^{(k)}$ denotes the graph Laplacian matrix at the $k^{\text{th}}$ iteration, $X_k^N$ and $X_k^\tau$ are the model parameter matrix of the adjacent best workers and adjacent maximum degree workers at the $k^{\text{th}}$ iteration. Assume $X_k^N = X_k A_k^N$, $X_k^\tau = X_k A_k^\tau$. Since every row in $X_k^N$ and $X_k^\tau$ is also a row of the model parameter matrix $X_k$, we could conclude that the transformation matrices $A_k^N$ and $A_k^\tau$ must be the left stochastic matrices.

AL-DSGD switches between $n$ communication graphs $\{G_{(i)}\}_{i=1}^n$. Let $\{L_{(i),j}\}_{j=1}^m$ be the Laplacian matrices set as matching decomposition of graph $G_{(i)}$. Led by MATCHA approach, to each matching of $L_{(i),j}$ to graph $G_{(i)}$ we assign an independent Bernoulli random variable $B_{(i),j}$ with probability $p_{(i),j}$ based on the communication budget $c_b$. Then the graph Laplacian matrix at the $k^{\text{th}}$ iteration $L^{(k)}$ can be written as: $\sum_{j=1}^m B_{(1),j}^{(k)} L_{(1),j}$(if $k \bmod n = 1$), $\sum_{j=1}^m B_{(2),j}^{(k)} L_{(2),j}$(if $k \bmod n = 2$),..., $\sum_{j=1}^m B_{(n),j}^{(k)} L_{(n),j}$ (if $k \bmod n = 0$). The next theorem captures the convergence of the AL-DSGD algorithm.

**Theorem 1.** *Let $\{L^{(k)}\}$ denote the sequence of Laplacian matrix generated by AL-DSGD algorithm with arbitrary communication budget $c_b > 0$ for the dynamic communication graph set $\{G_{(i)}\}_{i=1}^n$. Let the mixing matrix $\widetilde{W}^{(k)}$ be defined as in Equation 4). There exists a range of $\alpha$ and a range of average parameters $\omega_N = \omega_\tau \in (0, \omega(\alpha))$, whose bound is dictated by $\alpha$, such that the spectral norm $\rho = \max\left\{\mathbb{E}\left[\widetilde{W}^{(k)}(I-J)\widetilde{W}^{(k)\intercal}\right], \mathbb{E}\left[\widetilde{W}^{(k)}\widetilde{W}^{(k)\intercal}\right]\right\} < 1$, where $J = \mathbf{1}\mathbf{1}^\intercal/m$.*

Theorem 1 states that for arbitrary communication budget $c_b$ there exists some $\alpha$, $\omega_N$ and $\omega_\tau$ such that the spectral norm $\rho < 1$, which is a necessary condition for AL-DSGD to converge.

### 5.2 Convergence guarantee

This section provides the convergence guarantee for the proposed AL-DSGD algorithm. We define the average iterate as $\overline{x}^{(k)} = \frac{1}{m}\sum_{i=1}^m x_i^{(k)}$ and the minimum of the loss function as $F^*$. This section demonstrates that the averaged gradient norm $\frac{1}{K}\sum_{k=1}^K \mathbb{E}[\nabla F(\overline{x}^{(k)})]$ converges to zero with sub-linear convergence rate.

We assume that the loss function $F(x) = \sum_{i=1}^m F_i(x)$ satisfy the following conditions:

(1) *Lipschitz continuous:* $F_i(x) - F_i(y) \le \beta x - y$

(2) *Lipschitz gradient:* $\nabla F_i(x) - \nabla F_i(y) \le Lx - y$

(3) *Unbiased gradient:* $\mathbb{E}_{\xi_i}[g_i(x_k;\xi_i)] = \nabla F_i(x)$

(4) *Bounded variance:* $\mathbb{E}_{\xi_i}[g_i(x_k;\xi_i) - \nabla F_i(x)^2] \le \sigma^2$

(5) *Unified gradient:* $\mathbb{E}_{\xi_i}[\nabla F_i(x) - \nabla F(x)^2] \le \zeta^2$

(6) *Bounded domain:* $\max\{\|x_{k,i} - x_{k,i}^N\|, \|x_{k,i} - x_{k,i}^\tau\|\} \le \Delta^2$.

**Theorem 2.** *Suppose all local workers are initialized with $\overline{x}^{(1)} = 0$ and $\{\widetilde{W}^{(k)}\}_{k=1}^K$ is an i.i.d matrix sequence generated by AL-DSGD algorithm which satisfies the spectral norm $\rho < 1$ ($\rho$ is defined in Section 5.1). Under Assumption 5.2, if $\lambda = 2\lambda_N = 2\lambda_\tau$ and $(1-\alpha)(1-\omega)\gamma L \le \min\{1, (\sqrt{\rho^{-1}} - 1)\}$, then after K iterations:*

$$\begin{aligned} \frac{1}{K}\sum_{i=1}^{K} \mathbb{E}\left[\nabla F(\overline{x}_k)^2\right] \le &\frac{8(F(\overline{x}_1) - F^*)}{\eta K} + \frac{8M}{\eta} \\ &+ \frac{8\eta^2 L^2 \rho}{1-\sqrt{\rho}}\left(\frac{m\sigma^2 + \lambda^2\Delta^2}{m(1+\sqrt{\rho})} + \frac{3\zeta^2}{1-\sqrt{\rho}}\right), \end{aligned}$$

*where $\eta = (1-\alpha)(1-\omega)\gamma$ and $M = \frac{\eta^2 L\sigma^2}{2m} + \lambda\eta\beta\Delta + \lambda\eta^2 L\beta\Delta + \frac{\lambda^2\eta^2 L\Delta^2}{2}$. When setting $\lambda = \sqrt{\frac{m}{K}}$, $\gamma = \sqrt{\frac{m}{(1-\omega)(1-\alpha)K}}$, we obtain sub-linear convergence rate.*

Note that all assumptions in Assumption 5.2 are commonly used assumptions for decentralized distributed optimization [29, 41, 26]. $(1-\alpha)(1-\omega)L \le \min\{1, (\sqrt{\rho^{-1}} - 1)\}$ is a weak assumption on the learning rate and resembles similar assumption in Theorem 2 in [41]. Note that we give an exact value for the upper-bound on $\rho$ in supplementary material [19] Appendix B, which implies that under certain choices of $\alpha$, $\omega_N$, and $\omega_\tau$, $\rho$ could be much smaller than 1 and the right-hand side of the bound is therefore not approaching 0. Moreover, Assumption 5.2 (1) guarantees the Lipschitz constant for the loss objective function, and constructing learning rate based on the Lipschitz constant is widely used in many convergence proofs [44, 32, 45].

## 6 Experimental results

This section is devoted to the empirical evaluation of the proposed AL-DSGD scheme.

**Datasets and models:** In our experiments, we employ ResNet-50 and Wide ResNet models. The models are trained on CIFAR-10 and CIFAR-100 [25]. The same architectures and data sets were used by our competitor methods, MATCHA and D-PSGD. The training data set is randomly and evenly partitioned over a network of workers (each worker sees all the classes and the number of samples per classes are

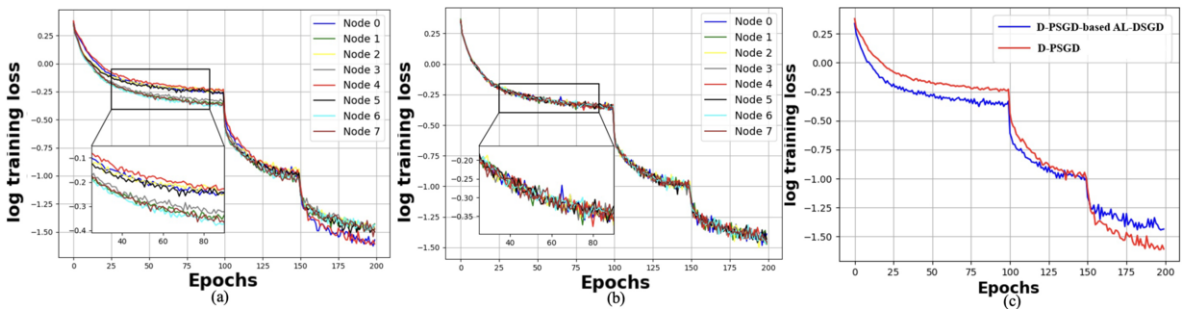


**Figure 6.** Training loss behavior for ResNet-50 trained on CIFAR-10. Optimization schemes: (a) D-PSGD, (b) D-PSGD-based AL-DSGD (c): Comparison between worst performing workers from a and b.

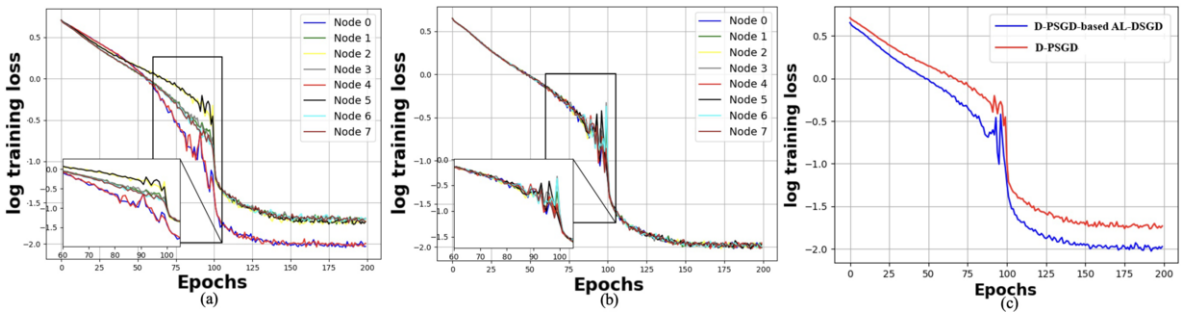


**Figure 7.** Training loss behavior for WideResNet trained on CIFAR-100. Optimization schemes: (a) D-PSGD, (b) D-PSGD-based AL-DSGD (c): Comparison between worst performing workers from a and b.

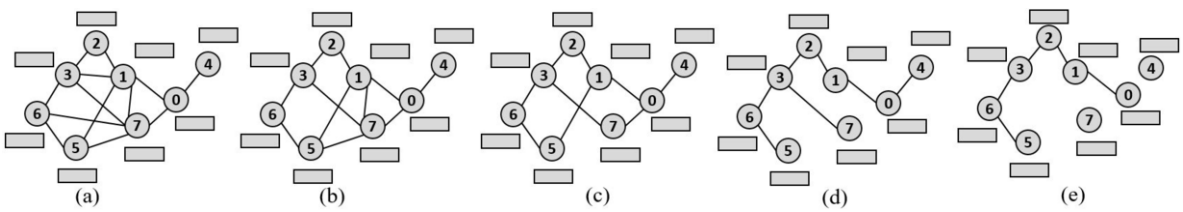


**Figure 8.** (a) Graph with total degree $D = 13, d = 100\%$. (b) Graph with reduced degree $d = 84.6\%, D = 11$, (c) Graph with reduced degree $d = 69.2\%, D = 9$, (d) Graph with reduced degree $d = 53.8\%, D = 7$, (e) Graph with reduced degree $d = 38.5\%, D = 5$.

the same across all the workers). In the decentralized synchronous setting, a pre-round barrier addresses computational speed variations (straggler issue) caused by hardware and data sampling differences. Slower workers naturally wait for faster ones to complete training before synchronization. This aligns with our baselines [41, 29].

**Machines/Clusters:** All the implementations are done in PyTorch and OpenMPI within mpi4py. We conduct experiments on a HPC cluster with 100Gbit/s network. In all of our experiments, we use RTX8000 GPU as workers.

**AL-DSGD and Competitors:** We implemented the proposed AL-DSGD with pulling coefficients $\lambda_N = 0.1$ and $\lambda_\tau = 0.1$. We set model weights coefficients to $w_N = 0.1$ and $w_\tau = 0.1$. The pulling coefficients and the model weights are fine-tuned for the ALD-SGD-based D-PSGD with ResNet-50 trained on CIFAR-10 and then used for other experiments. We compared our algorithm with the D-PSGD and MATCHA methods, where in case of MATCHA the communication budget $c_b$ was set to $c_b = 0.5$, as recommended by the authors.

**Implementations:** All algorithms are trained for a sufficiently long time until convergence or onset of over-fitting. The learning rate is fine-tuned for the D-PSGD baseline and then used for MATCHA and AL-DSGD algorithm. The initial learning rate is 0.4 and reduced by a factor of 10 after 100 and 150 epochs. The batch size per worker node is 128. We randomly selected representative examples of results from multiple runs under the same settings. With small standard deviation, the conclusion of experiments remain unchanged from multiple runs.

## 6.1 *Convergence and performance*

The results from training models with D-PSGD and D-PSGD-based AL-DSGD (AL-DSGD on top of D-PSGD) are in Table 2. MATCHA and MATCHA-based AL-DSGD results are in supplementary material [19], Appendix F, Table 10. Training loss, shown in Figure 6 and 7, assesses convergence better due to unstable test loss. AL-DSGD reduces variance between nodes and speeds up convergence for the worst-performing node.

***In summary, AL-DSGD has enhanced the test accuracy for both the average and worst-performing models.*** Tables 2 and Table 10 in supplementary material [19] reveal AL-DSGD's superior

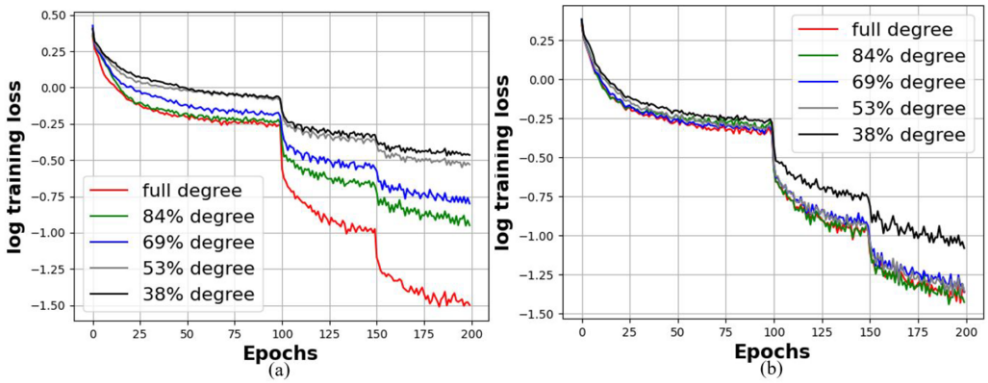


**Figure 9.** ResNet-50 model trained on CIFAR-10. Performance of a) D-PSGD and b) AL-DSGD-based D-PSGD with different topology degrees.

| | CIFAR-10/ResNet-50 | | CIFAR-100/WideResNet | |
|---|---|---|---|---|
| **Node** | **D-PSGD** | **AL-DSGD** | **D-PSGD** | **AL-DSGD** |
| 0 | **87.95**↓ | 93.68 | **59.45**↓ | 76.18 |
| 1 | 92.11 | 93.72 | 74.70 | 76.35 |
| 2 | 92.21 | 93.55 | 74.65 | 76.10 |
| 3 | 92.36 | 93.87 | 74.16 | 76.36 |
| 4 | **87.86**↓ | 93.83 | **59.63**↓ | 76.51 |
| 5 | 92.25 | 93.48 | 74.62 | 76.34 |
| 6 | 92.38 | 93.65 | 74.59 | 76.39 |
| 7 | 92.32 | 93.62 | 74.57 | 76.30 |
| **AVG** | 91.18 | **93.68** | 70.79 | **76.31** |

**Table 2.** Test accuracy obtained with D-PSGD and D-PSGD-based AL-DSGD for ResNet-50 model trained on CIFAR-10 and WideResNet model trained on CIFAR-100.

generalization over other methods. For the case of D-PSGD, the test accuracy has increased by resp. $5.8\%$ and $16.7\%$ in the worst-performance worker and by resp. $2.1\%$ and $5.5\%$ in the final averaged model for CIFAR-10/ResNet50 and CIFAR-100/WideResNet tasks, respectively, when putting AL-DSGD on the top of D-PSGD. For the case of MATCHA, even though the AL-DSGD does not dramatically increase the baseline performance for CIFAR-10/ResNet50 task because the model is relatively simple, it strongly outperforms the baseline on more complicated CIFAR100/WideResNet task. As shown in Figure 13 in the supplementary material [19] Appendix F, AL-DSGD demonstrates more stable (i.e., smaller discrepancies between nodes) and faster convergence compared to MATCHA.

We would like to further emphasize that, except for converging to a better optimum, another significant advantage of our AL-DSGD algorithm is that it is much more robust to imbalanced and sparse topology, as will be discussed in the following section.

### *6.2 Communication*

In this subsection, we evaluate algorithm performance using ResNet-50 on CIFAR-10 in a communication-constrained environment. ***The experiments overall demonstrate that AL-DSGD has enhanced test accuracy in scenarios involving either imbalanced or sparse topologies.*** Our approach relies on a dynamic communication graph with three Laplacian matrices (Figure 5). The results are shown here, and the results using two Laplacian matrices are in supplementary material [19] Appendix E. Tables 2 and Table 10 in supplementary material [19] Appendix D demonstrate that D-PSGD-based AL-DSGD and MATCHA-based AL-DSGD are more robust to imbalanced topologies. To further evaluate AL-DSGD in communication-limited environments, we gradually reduce the communication graph's degree to simulate sparse topology (Figure 8) and compare AL-DSGD's performance with D-PSGD (Figure 9). AL-DSGD remains stable and robust to sparse topologies, as its performance does not significantly decrease until the degree is reduced to 38%, while D-PSGD performs poorly even when the degree is only decreased to 84%.

| | TEST ACCURACY | | | | |
|---|---|---|---|---|---|
| **Algorithm** | D=13 | D=11 | D=9 | D=7 | D=5 |
| **D-PSGD** | 91.18 | 91.21 | 90.98 | 90.26 | 89.59 |
| **AL-DSGD-based D-PSGD** | **93.68** | **93.59** | **93.58** | **93.30** | **92.32** |
| **MATCHA** | 93.65 | **93.51** | 93.24 | 92.86 | 91.14 |
| **AL-DSGD-based MATCHA** | **93.94** | 93.33 | **93.49** | **93.30** | **92.75** |

**Table 3.** The performance of D-PSGD, MATCHA, AL-DSGD-based D-PSGD, and AL-DSGD-based MATCHA for topolgies with different total degrees $D$. Results were obtained for CIFAR10 and ResNet-50.

Finally, Table 3 includes the comparison of D-PSGD, MATCHA, and AL-DSGD. We applied AL-DSGD on the top of both the D-PSGD and MATCHA baselines and compared the results. Table 3 further confirms the claim that the AL-DSGD algorithm is highly robust to sparse topologies, as it consistently achieves better test accuracy compared to the baseline algorithms, D-PSGD and MATCHA, for nearly all the cases, particularly in sparse topology scenarios. More details can be found in supplementary material [19] Appendix D & F.

## 7 Conclusions

This paper introduces Adjacent Leader Decentralized Gradient Descent (AL-DSGD), a novel decentralized distributed SGD algorithm. AL-DSGD assigns weights to neighboring learners based on their performance and degree for averaging and integrates corrective forces from best-performing and highest-degree neighbors during training. By employing a dynamic communication graph, AL-DSGD excels in communication-constrained scenarios, including imbalanced and sparse topologies. Theoretical proof of algorithm convergence is provided. Experimental results on various datasets and deep architectures demonstrate that AL-DSGD accelerates and stabilizes convergence of decentralized state-of-the-art techniques, improving test performance, especially in communication-constrained environments.

## References


[1] S. A. Alghunaim and K. Yuan. A unified and refined convergence analysis for non-convex decentralized learning. *IEEE Transactions on Signal Processing*, 70:3264–3279, 2022.

[2] I. Benjamini, G. Kozma, and N. Wormald. The mixing time of the giant component of a random graph. *Random Structures & Algorithms*, 45(3): 383–407, 2014.

[3] A. S. Berahas, R. Bollapragada, N. S. Keskar, and E. Wei. Balancing communication and computation in distributed optimization. *IEEE Transactions on Automatic Control*, 64(8):3141–3155, 2018.

[4] M. Blot, D. Picard, M. Cord, and N. Thome. Gossip training for deep learning. *arXiv preprint arXiv:1611.09726*, 2016.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Y. Chen, K. Yuan, Y. Zhang, P. Pan, Y. Xu, and W. Yin. Accelerating gossip sgd with periodic global averaging. In *International Conference on Machine Learning*, pages 1791–1802. PMLR, 2021.

[7] Y.-T. Chow, W. Shi, T. Wu, and W. Yin. Expander graph and communication-efficient decentralized optimization. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 1715–1720. IEEE, 2016.

[8] H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 37–48, 2014.

[9] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[11] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1), 2012.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2011.

[15] S. Dutta, V. Cadambe, and P. Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. *Advances In Neural Information Processing Systems*, 29, 2016.

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

[17] H. He and P. Dube. Accelerating parallel stochastic gradient descent via non-blocking mini-batches. *arXiv preprint arXiv:2211.00889*, 2022.

[18] H. He and P. Dube. Rcd-sgd: Resource-constrained distributed sgd in heterogeneous environment via submodular partitioning. *arXiv preprint arXiv:2211.00839*, 2022.

[19] H. He, J. Wang, and A. Choromanska. Adjacent leader decentralized stochastic gradient descent. *arXiv preprint arXiv:2405.11389*, 2024.

[20] K. Huang and S. Pu. Improving the transient times for distributed stochastic gradient methods. *IEEE Transactions on Automatic Control*, 2022.

[21] D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar. Convergence rates for distributed stochastic optimization over random networks. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4238–4245. IEEE, 2018.

[22] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581*, 2016.

[23] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[24] L. Kong, T. Lin, A. Koloskova, M. Jaggi, and S. Stich. Consensus control for decentralized deep learning. In *International Conference on Machine Learning*, pages 5686–5696. PMLR, 2021.

[25] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] H. Li, C. Fang, and Z. Lin. Convergence rates analysis of the quadratic penalty method and its applications to decentralized distributed optimization. *arXiv preprint arXiv:1711.10802*, 2017.

[27] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27, 2014.

[28] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[29] 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. *Advances in Neural Information Processing Systems*, 30, 2017.

[30] X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[31] M. Liu, W. Zhang, Y. Mroueh, X. Cui, J. Ross, T. Yang, and P. Das. A decentralized parallel algorithm for training generative adversarial nets. *Advances in Neural Information Processing Systems*, 33:11056–11070, 2020.

[32] Z. Mhammedi, W. M. Koolen, and T. Van Erven. Lipschitz adaptivity with multiple learning rates in online learning. In *Conference on Learning Theory*, pages 2490–2511. PMLR, 2019.

[33] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1): 48–61, 2009.

[34] A. Nedić, A. Olshevsky, and M. G. Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018.

[35] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[36] K. Scaman, F. Bach, S. Bubeck, L. Massoulié, and Y. T. Lee. Optimal algorithms for non-smooth distributed optimization in networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[37] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu. $D^2$: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856. PMLR, 2018.

[38] Y. Teng, W. Gao, F. Chalus, A. E. Choromanska, D. Goldfarb, and A. Weller. Leader stochastic gradient descent for distributed training of deep learning models. *Advances in Neural Information Processing Systems*, 32, 2019.

[39] Z. J. Towfic, J. Chen, and A. H. Sayed. Excess-risk of distributed stochastic learners. *IEEE Transactions on Information Theory*, 62(10): 5753–5785, 2016.

[40] J. Wang and G. Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *The Journal of Machine Learning Research*, 22(1):9709–9758, 2021.

[41] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar. Matcha: Speeding up decentralized sgd via matching decomposition sampling. In *2019 Sixth Indian Control Conference (ICC)*, pages 299–300. IEEE, 2019.

[42] J. Wang, V. Tantia, N. Ballas, and M. Rabbat. Slowmo: Improving communication-efficient distributed sgd with slow momentum. *arXiv preprint arXiv:1910.00643*, 2019.

[43] J. Wang, A. K. Sahu, G. Joshi, and S. Kar. Matcha: A matching-based link scheduling strategy to speed up distributed optimization. *IEEE Transactions on Signal Processing*, 70:5208–5221, 2022.

[44] X. Wu, R. Ward, and L. Bottou. Wngrad: Learn the learning rate in gradient descent. *arXiv preprint arXiv:1803.02865*, 2018.

[45] R. Yedida, S. Saha, and T. Prashanth. Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence. *Applied Intelligence*, 51:1460–1478, 2021.

[46] K. Yuan, Q. Ling, and W. Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854, 2016.

[47] K. Yuan, S. A. Alghunaim, B. Ying, and A. H. Sayed. On the influence of bias-correction on distributed stochastic optimization. *IEEE Transactions on Signal Processing*, 68:4352–4367, 2020.

[48] K. Yuan, S. A. Alghunaim, and X. Huang. Removing data heterogeneity influence enhances network topology dependence of decentralized sgd. *arXiv preprint arXiv:2105.08023*, 2021.

[49] J. Zeng and W. Yin. On nonconvex decentralized gradient descent. *IEEE Transactions on signal processing*, 66(11):2834–2848, 2018.

[50] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. *Advances in neural information processing systems*, 28, 2015.