Distributed Learning in the Non-Convex World: From Batch to Streaming Data, and Beyond

Tsung-Hui Chang, Mingyi Hong, Hoi-To Wai, Xinwei Zhang, Songtao Lu

Abstract

Distributed learning has become a critical enabler of the massively connected world envisioned by many. This article discusses four key elements of scalable distributed processing and real-time intelligence — problems, data, communication and computation. Our aim is to provide a fresh and unique perspective about how these elements should work together in an effective and coherent manner. In particular, we provide a selective review about the recent techniques developed for optimizing *non-convex* models (i.e., problem classes), processing *batch and streaming* data (i.e., data types), over the networks in a distributed manner (i.e., communication and computation paradigm). We describe the intuitions and connections behind a core set of popular distributed algorithms, emphasizing how to trade off between computation and communication costs. Practical issues and future research directions will also be discussed.

I. Introduction

We are living in a highly connected world, and it will become exponentially more connected in a decade. By 2030, there will be more than 125 billion interconnected smart devices, creating a massive network of intelligent appliances, cars, gadgets and tools (https://developer.ibm.com/articles/se-iot-security/). These devices collect a huge amount of real-time data, perform complex computational tasks, and provide vital services which significantly improve our lives and enrich our collective productivity.

There are four key elements that enable scalable distributed processing and real-time intelligence in a massively connected world — problems, data, communication and computation. These elements are closely tied with each other as illustrated in Fig. 1. For example, without a meaningful machine learning (ML) problem, crunching large amounts of data using massive computational resources rarely lead to

THC, MH, HTW are ordered alphabetically, and contributed equally. THC is with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China. MH, SL, XZ are with the ECE Department, University of Minnesota, MN, USA. HTW is with the Department of SEEM, The Chinese University of Hong Kong, Hong Kong SAR, China. HTWs work is supported by the CUHK Direct Grant #4055113, MH, SL and XZ are supported in part by NSF grants CMMI-172775, CIF-1910385 and by an ARO grant 73202-CS.

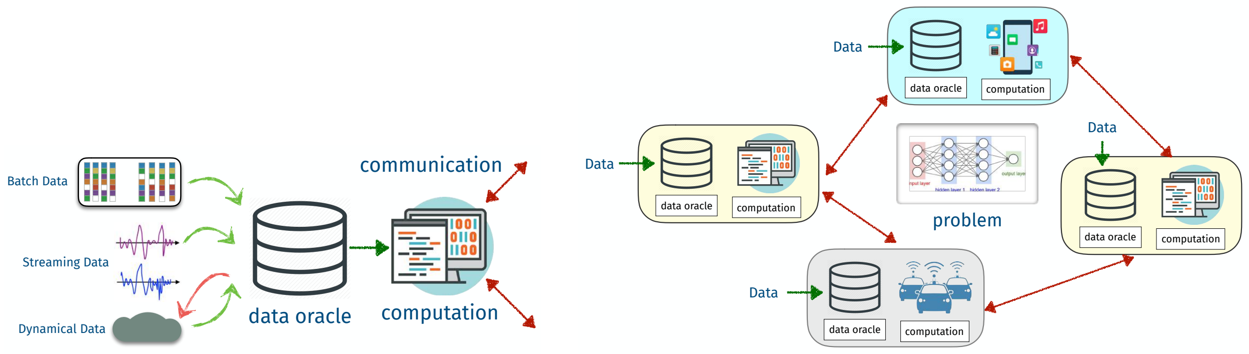


Fig. 1: Overview of the key elements in distributed learning. (Left) Flow between different elements at a single agent: **data** (e.g., images) is taken from diverse types through an *oracle*, **processed locally** before **communicating** with other agents, the final goal is to tackle a non-convex learning **problem**. (Right) Distributed learning on a network of agents.

any actionable intelligence. Similarly, despite all its sophisticated design and nice interpretation from neural sciences, modern neural networks may not be so successful without highly efficient computation methods. The *overarching goal* of this selective review is to provide a fresh and unique perspective about how these elements should work together in the most effective and coherent manner, as to realize scalable processing, real-time intelligence, and ultimately contribute to the vision of a highly smart and connected world. Some key aspects of these elements to be taken into consideration are outlined below:

Non-Convexity. For many emerging distributed applications, the problems to be solved by distributed nodes will be highly complicated. For instance, in distributed ML, the nodes (e.g., mobile devices) jointly learn a model based on the local data (e.g., images on each device). To accurately represent the local data, the nodes are often required to use *non-convex* loss functions, such as the composition of multiple nonlinear activation functions in collaborative deep learning [1]–[3].

Data Acquisition Processes. One of the main reasons behind the recent success of ML is the ability to process data at scale. This not only means that one can process large volumes of data quickly (i.e., dealing with *batch* data), but more importantly, it requires capability of dealing with *streaming* data. There is an urgent need, and hence a growing research, to deal with the massive amount of streaming data from online review platforms (e.g., Amazon), social network (e.g., Facebook), etc.

Distributed Processing. The growing network size, the increased amount of the distributed data, and the requirements for real-time response often make traditional centralized processing not viable. For example, self-driving cars should be carefully coordinated when meeting at an intersection, but since every such vehicle can generate up to 40 Gbit of data (e.g., from LIDAR and cameras) per second – an amount that overwhelms the fastest cellular network – it is impossible to pool the entirety of data for real-time central coordination. This and other examples, from small and ordinary (e.g. coordinating smart appliance in home) to large and vitally important (e.g., national power distribution), show how paramount fast distributed processing will be to our collective well-being, productivity and prosperity.

This paper is a *selective review* about recent advances on distributed algorithms. Unlike existing articles [4]–[6], this paper is centered around *non-convex* optimization and learning problems. Our focus is to reveal connections and design insights about a core set of first-order algorithms, while highlighting the interplay between problem, data, computation, and communication. We hope that the algorithm connections identified in this article will assist the readers in comparing theoretical and practical properties between algorithms, and will help translate new features and theoretical advances developed for one type of algorithms to other "equivalent" types, without risking to "reinventing the wheel".

We will start with a generic model of distributed optimization, taking into consideration diverse data types and non-convex models (Sec. II). We will then review state-of-the-art algorithms that deal with batch/streaming data, make useful connections between them (Sec. III), and discuss practical issues in implementing these algorithms (Sec. IV). Finally, we discuss future research directions (Sec. V).

II. Problems and Data Models

Problem Class. Consider n inter-connected agents. The network connecting these agents is represented by a (directed or undirected) graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., n\}$ is the set of agents and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of communication links between agents. The goal for agents is to find a solution $\boldsymbol{\vartheta}^\star := (\boldsymbol{\theta}_1^\star, ..., \boldsymbol{\theta}_n^\star)$ which tackles the *non-convex* optimization problem:

$$\min_{\boldsymbol{\theta}_i \in \mathbb{R}^d, \forall i} \; \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\theta}_i) \;\; \text{s.t.} \;\; (\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n) \in \mathcal{H} := \big\{ \boldsymbol{\theta}_i \in \mathbb{R}^d, \; i = 1, ..., n : \boldsymbol{\theta}_i = \boldsymbol{\theta}_j, \; \forall \; (i,j) \in \mathcal{E} \big\}. \quad (1)$$

We define $f(\boldsymbol{\vartheta}) := \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\theta}_i)$, where $f : \mathcal{H} \to \mathbb{R} \cup \{\infty\}$ is a (possibly non-convex) cost function of the ith agent. Problem (1) contains a coupling constraint that enforces *consensus*. When G is undirected and connected, we have $\boldsymbol{\theta}_1 = \cdots = \boldsymbol{\theta}_n$, and an optimal solution to (1) is a minimizer to the finite sum $\sum_{i=1}^{n} f_i(\boldsymbol{\theta})$. The consensus formulation motivates a *decentralized* approach to finding high-quality solutions, in the sense that each agent i can only access its local cost function f_i and process its local data together with messages exchanged from its neighbors.

Without assuming convexity for (1), one cannot hope to find an optimal solution using a reasonable amount of effort, as solving a non-convex problem is in general NP-hard [7]. Instead, we resort to finding *stationary, consensual solutions* whose gradients are small and the variables are in consensus.

Formally, let $\epsilon \geq 0$, we say that $\boldsymbol{\vartheta} = (\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n)$ is an ϵ-stationary solution to (1) if

$$\mathsf{Gap}(\boldsymbol{\vartheta}) := \left\| n^{-1} \sum_{j=1}^n \nabla f_j \big(\bar{\boldsymbol{\theta}} \big) \right\|^2 + \sum_{j=1}^n \| \boldsymbol{\theta}_j - \bar{\boldsymbol{\theta}} \|^2 \leq \epsilon, \text{ where } \bar{\boldsymbol{\theta}} := n^{-1} \sum_{i=1}^n \boldsymbol{\theta}_i. \tag{2}$$

Below we summarize two commonly used conditions when approaching problem (1).

Assumption 1. (a) *The graph G is undirected and connected.* (b) *For $i = 1, ..., n$, the cost function $f_i(\boldsymbol{\theta})$ is L-smooth, satisfying the following condition:*

$$\|\nabla f_i(\boldsymbol{\theta}) - \nabla f_i(\boldsymbol{\theta}')\| \leq L \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \ \forall \ \boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d. \tag{3}$$

Further, $f : \mathcal{H} \to \mathbb{R} \cup \{\infty\}$, that is, the average function f is lower bounded over the domain $\mathcal{H}$.

Assumption 1, together with (1), describes a general setup for distributed learning problems. Our goal is to find a stationary, consensual solution satisfying (2). We remark that several recent works [8]–[12] have analyzed the more powerful forms of convergence, such as to second-order stationary solutions or global optimal solutions. However, establishing these results requires additional assumptions, which further restrict the problem class, so we shall omit discussing these works in detail due to space limitation.

Having fixed the problem class, a distributed learning system consists of a data acquisition and local processing step performed at a local agent, and a communication step to exchange information between agents. We summarize these key elements and their interactions in Fig. 1.

Data Model. We adopt a *local oracle model [denoted as $\mathsf{DO}_i(\boldsymbol{\theta}_i)$]* to describe how information about the cost function f_i is retrieved in distributed learning. As we shall focus on first-order algorithms in the sequel, the oracle $\mathsf{DO}_i(\boldsymbol{\theta}_i)$ is characterized as various estimates of the gradient $\nabla f_i(\boldsymbol{\theta}_i)$.

a) Batch Data: This is a classical setting where the entire local data set is available at *anytime*, also known as the *offline* learning setting. Denote $\xi_{i,1}, \xi_{i,2}, ..., \xi_{i,M_i}$ as the local dataset of agent i with M_i data samples. The local cost function and DO are given by the *finite sums* below:

$$f_i(\boldsymbol{\theta}_i) = M_i^{-1} \sum_{\ell=1}^{M_i} F_i(\boldsymbol{\theta}_i; \xi_{i,\ell}), \quad \mathsf{DO}_i(\boldsymbol{\theta}_i^t) = \nabla f_i(\boldsymbol{\theta}_i^t) \tag{4}$$

where $F_i(\boldsymbol{\theta}_i; \xi_{i,\ell})$ is the cost function corresponding to the (i, ℓ)th data.

b) Streaming Data: In this setting, the data are revealed in a *streaming* (or *online*) fashion. We first specify our cost function as a stochastic function $f_i(\boldsymbol{\theta}_i) = \mathbb{E}_{\xi_i \sim \pi_i(\cdot)} \big[F_i(\boldsymbol{\theta}_i; \xi_i) \big]$, where $\pi_i(\cdot)$ is a probability distribution of ξ_i. At each iteration t, querying the DO draws m_t independent and identically distributed (i.i.d.) samples for the learning task; thus,

$$\mathsf{DO}_i(\boldsymbol{\theta}_i^t) = m_t^{-1} \sum_{\ell=1}^{m_t} \nabla F_i(\boldsymbol{\theta}_i^t; \xi_{i,\ell}^{t+1}) \text{ where } \xi_{i,\ell}^{t+1} \sim \pi_i(\cdot), \ \ell = 1, ..., m_t, \tag{5}$$

which is an unbiased estimate of gradient, i.e., $\mathbb{E}_{\xi \sim \pi_i(\cdot)}[\mathsf{DO}_i(\boldsymbol{\theta})] = \nabla f_i(\boldsymbol{\theta})$. Moreover,

Assumption 2. *Consider the DO* (5) *and random samples ξ drawn i.i.d. from $\pi_i(\cdot)$. Assume*

$$\mathbb{E}_{\xi \sim \pi_i(\cdot)}[\|\mathsf{DO}_i(\boldsymbol{\theta}) - \nabla f_i(\boldsymbol{\theta})\|^2] \leq \sigma < \infty, \; i = 1, ..., n, \; \forall \; \boldsymbol{\theta} \in \mathbb{R}^d. \tag{6}$$

In other words, the random variable $\nabla f_i(\xi)$ has a bounded variance.

A related setting involves a large but fixed dataset ($M_i \gg 1$) at agent i, denoted by $\{\xi_{i,1}, ..., \xi_{i,M_i}\}$ and f_i is given by (4). Accessing the full dataset entails an undesirable $\mathcal{O}(M_i)$ *computation* complexity. As a remedy, we can draw at each iteration a small batch of random samples ($m \ll M_i$) *uniformly* from the large dataset. This results in a DO akin to (5).

A. Examples and Challenges

We conclude this section by listing a few popular examples of non-convex learning problems and how they fit into the described models above. Moreover, we discuss the challenges with non-convex distributed learning, which motivate many algorithms reviewed in this paper.

Example 1. (Binary Classifier Training with Neural Network) *For each $i \in \{1, ..., n\}$, suppose that a stream of training data $\xi_{i,1}^t, \xi_{i,2}^t, ...$ is available at the ith agent, where $\xi_{i,j}^t = (\boldsymbol{x}_{i,j}^t, y_{i,j}^t)$ is a tuple containing the feature $\boldsymbol{x}_{i,j}^t \in \mathbb{R}^m$ and label $y_{i,j}^t \in \{0, 1\}$. Let $\boldsymbol{\theta} = (\mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)})$ be the parameters of an L-layer neural network, we consider the models* (1) *with the following logistic loss:*

$$F_i(\boldsymbol{\theta}; \xi_{i,j}^t) = (1 - y_{i,j}^t) \log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,j}^t)) + y_{i,j}^t \log h_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,j}^t), \tag{7}$$

where $h_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,j}^t)$ is the sigmoid function $(1 + g(\boldsymbol{x}_{i,j}^t; \boldsymbol{\theta}))^{-1}$ such that $g(\boldsymbol{x}_{i,j}^t; \boldsymbol{\theta})$ is the soft-max output of the last layer of the neural network with $\boldsymbol{x}_{i,j}^t$ being the input. The hidden layer of the neural network may be defined as $\boldsymbol{g}^{(\ell+1)} = u(\mathbf{W}^{(\ell+1)} \boldsymbol{g}^{(\ell)})$ for $\ell = 0, ..., L-1$, where $u(\cdot)$ is an activation function and $\boldsymbol{g}^{(0)} = \boldsymbol{x}_{i,j}^t$. The goal of (1) *is to find a set of optimal parameters of a neural network, taking into account the (potentially heterogeneous) data received at all agents. Here, the loss function $f_i(\boldsymbol{\theta})$ is non-convex but satisfies **Assumption 1**, and the DO follows the **streaming data** model.*

Example 2. (Matrix Factorization) *The ith agent has a fixed set of M_i samples where the ℓth sample is denoted as $\xi_{i,\ell} = \boldsymbol{x}_{i,\ell} \in \mathbb{R}^{m_1}$. The data received at the agents can be encoded using the columns of a* dictionary matrix $\boldsymbol{\Phi} \in \mathbb{R}^{m_1 \times m_2}$, i.e., $\boldsymbol{x}_{i,\ell} \approx \boldsymbol{\Phi} \boldsymbol{y}_{i,\ell}$. *The goal is to learn a factorization with the* dictionary *$\boldsymbol{\Phi}$ and codes $\boldsymbol{Y}_i = (\boldsymbol{y}_{i,1} \; \cdots \; \boldsymbol{y}_{i,M_i})$. Let $\boldsymbol{X}_i = (\boldsymbol{x}_{i,1} \; \cdots \; \boldsymbol{x}_{i,M_i})$ be the data. The learning problem is:*

$$\min_{\boldsymbol{\Phi}, \boldsymbol{Y}_i, i=1,...,n} \; \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{X}_i - \boldsymbol{\Phi} \boldsymbol{Y}_i\|_{\rm F}^2 \quad \text{s.t.} \quad \boldsymbol{\Phi} \in \mathsf{A}, \; \boldsymbol{Y}_i \in \mathsf{Y}_i, \; i = 1, ..., n, \tag{8}$$

where A, Y_i represent some constraints on the dictionary and codes to ensure identifiability. An interesting aspect of (8) *is that the problem optimizes a* common *variable* $\boldsymbol{\Phi}$ *and a* private *variable* $\boldsymbol{Y}_i$*; in particular, the corresponding local cost* $f_i(\boldsymbol{\Phi}) = \min_{\boldsymbol{Y}_i \in \mathsf{Y}_i} \|\boldsymbol{X}_i - \boldsymbol{\Phi}\boldsymbol{Y}_i\|_{\rm F}^2$*. The common variable is jointly decided by the data received at agents, while the private variables are nuisance parameters decided locally. Here, the loss function* $f_i(\boldsymbol{\theta})$ *satisfies* ***Assumption 1****, and the DO follows the* **batch data** *model.*

Handling non-convex distributed learning problems involves several unique challenges. First, directly applying algorithms developed for convex problems to the non-convex setting may lead to unexpected algorithm behaviors. To see this, consider a simple example as follows.

Example 3. *Consider* (1) *with* $d = 1$, $n = 2$ *agents connected via one edge. Let* $f_1(\theta_1) = \theta_1^2/2$, *and* $f_2(\theta_2) = -\theta_2^2/2$, *where* f_2 *is* non-convex. *Note that any* $\theta_1 = \theta_2$ *is an optimal solution to the consensus problem. However, applying the classical distributed gradient descent (DGD) method [13] (to be discussed in Sec. III-A), with a* constant step size $\gamma > 0$, *generates the following iterates:*

$$\boldsymbol{\theta}^{t+1} := \begin{pmatrix} \theta_1^{t+1} \\ \theta_2^{t+1} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \boldsymbol{\theta}^t - \gamma \begin{pmatrix} \theta_1^t \\ -\theta_2^t \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{2} - \gamma & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} + \gamma \end{pmatrix}}_{:=\boldsymbol{M}(\gamma)} \boldsymbol{\theta}^t \tag{9}$$

For all $\gamma > 0$, *the spectral radius of* $\boldsymbol{M}(\gamma)$ *is larger than one, so the above iteration always diverges. On the contrary, it can be verified that DGD converges linearly to a solution satisfying* $\theta_1 = \theta_2$ *with any positive step sizes if we change the objective functions to* $f_1(\theta_1) = f_2(\theta_2) = 0$. *Generally, if the problem is convex, DGD converges to a neighborhood of the optimal solution when small constant step sizes are used. This is in contrast to* (9) *which diverges regardless (as long as the step size is a constant).*

Second, it is challenging to deal with *heterogeneous data* where the data distribution of the agents are significantly different. This is because the local update directions can be different compared with the information communicated from the neighbors. Considering Example 3 again, the divergence of DGD can be attributed to the fact that the local functions have different local data, leading to $\nabla f_1(\theta) = -\nabla f_2(\theta)$.

Other practical challenges include how to implement distributed algorithms such that they scale to large networks and model sizes. Moreover, an effective distributed algorithm should jointly design the communication and computation protocols. Addressing these challenges will be the main focus next.

III. Balancing Communication and Computation in Distributed Learning

We study distributed algorithms for tackling problem (1). For simplicity, we assume scalar optimization variable, i.e., $d = 1$, throughout this section. Distributed algorithms require a balanced design for the

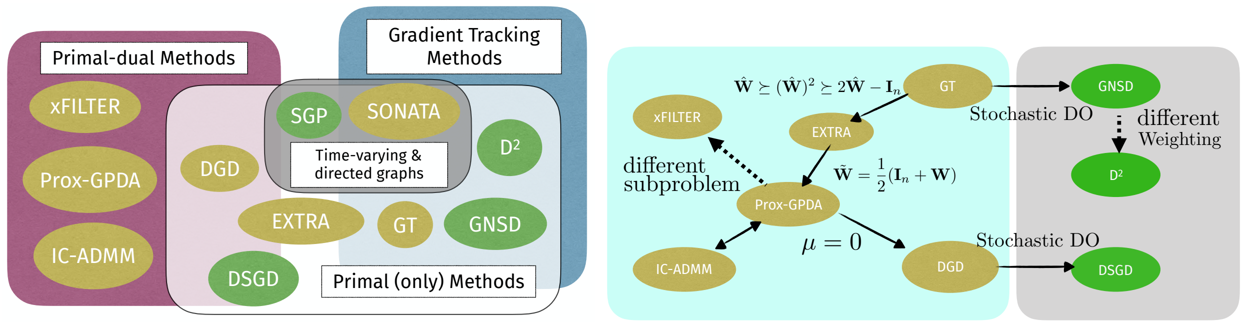


Fig. 2: An overview of distributed algorithms for non-convex learning. Orange (*resp.* green) patches refer to algorithms designed for batch (*resp.* streaming) data. Line with a single arrow indicates one algorithm can reduce to another; line with double arrows means algorithms are equivalent; dotted line indicates the algorithms are related (conditions given above the line).

computation and *communication* capability of a distributed learning system. This section shall delineate how existing algorithms overcome such challenge with different data oracles. In a nutshell, the *batch* data setting can be tackled using either a *primal-dual optimization* framework, or a family of *gradient tracking* methods; while the *streaming* data setting is commonly tackled by the distributed gradient descent or gradient tracking methods. A summary of the reviewed algorithms can be found in Fig. 2 (Left), and the connections between algorithms are illustrated in Fig. 2 (Right). Next, we review some basic concepts about distributed processing on networks.

Considering an undirected graph $G = (\mathcal{V}, \mathcal{E})$, we define its *degree matrix* as $\boldsymbol{D} := \text{diag}(d_1, ..., d_n)$, where d_i is the degree of node i. The *graph incidence matrix* $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{E}| \times n}$ has $A_{ei} = 1$, $A_{ej} = -1$ if $j > i$, $e = (i,j) \in \mathcal{E}$, and $A_{ek} = 0$ for all $k \in \mathcal{V} \setminus \{i,j\}$. Note that $\boldsymbol{A}^\top \boldsymbol{A} := \boldsymbol{L}_G \in \mathbb{R}^{n \times n}$ is the *graph Laplacian* matrix. Lastly, a *mixing matrix* $\boldsymbol{W}$ satisfies the following conditions:

$$\text{P1) null}\{\boldsymbol{I}_n - \boldsymbol{W}\} = \text{span}\{\boldsymbol{1}\};\ \text{P2)} -\boldsymbol{I}_n \prec \boldsymbol{W} \prec \boldsymbol{I}_n;\ \text{P3)}\ W_{ij} = 0 \text{ if } (i,j) \notin \mathcal{E}, W_{ij} > 0 \text{ o.w..} \tag{10}$$

For instance, the mixing matrix can be chosen as the doubly stochastic matrix:

$$W_{ij} = 1/d_{\mathsf{max}} \text{ if } (i,j) \in \mathcal{E}, \quad W_{ij} = 1 - d_i/d_{\mathsf{max}} \ \text{ if } i = j, \quad W_{ij} = 0 \text{ if } (i,j) \notin \mathcal{E}, \tag{11}$$

provided that the maximum degree $d_{\mathsf{max}} := \max_{i \in \mathcal{V}} d_i$ is available; see [14] for other designs. For any $\boldsymbol{\vartheta} \in \mathbb{R}^n$, we observe $\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\vartheta}$, $\boldsymbol{W}\boldsymbol{\vartheta}$ can be calculated via message exchange among neighboring agents.

The mixing and/or graph Laplacian matrix specifies the *communication* pattern in distributed learning. Since $\boldsymbol{W}^\infty = (1/n)\boldsymbol{1}\boldsymbol{1}^\top$, the mixing matrix allows one to compute the average distributively by repeatedly applying the mixing matrix. As the gradient $\frac{1}{n}\sum_{i=1}^n \nabla f_i(\theta)$ is the average of local gradients, the easiest way to derive a 'distributed' algorithm is to compute the exact average of gradient by applying $\boldsymbol{W}$ repeatedly. Such 'distributed' algorithm will behave exactly the same as the centralized gradient algorithm, and it may save *computation* (gradient or data oracle evaluation) with a faster convergence rate, yet the *communication* (message exchange) cost can be overwhelming.

A. Algorithms for Batch Data

In the batch data setting, the data oracle returns an exact gradient $\mathsf{DO}_i(\theta_i) = \nabla f_i(\theta_i)$ at the ith agent at anytime. This setting is typical with small-to-moderate dataset where gradient computation is cheap. To this end, the general design philosophy is to adopt techniques developed for *deterministic* first-order methods and specialize them to distributed learning considering the *communication* constraint.

Primal-Dual Methods. Let $\boldsymbol{\vartheta} = (\theta_1, \dots, \theta_n)^\top$ be the collection of local variables, we observe that the consensus constraint $\mathcal{H}$ can be rewritten as a set of linear equalities $\mathcal{H} = \{\boldsymbol{\vartheta} = (\theta_1, \dots, \theta_n)^\top \mid \boldsymbol{A}\boldsymbol{\vartheta} = \boldsymbol{0}\}$. It is then natural to consider the augmented Lagrangian (AL) of (1):

$$\mathcal{L}(\boldsymbol{\vartheta}, \boldsymbol{\mu}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta_i) + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\vartheta} + \frac{c}{2}\|\boldsymbol{A}\boldsymbol{\vartheta}\|^2, \tag{12}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{E}|}$ is the dual variable of the constraint $\boldsymbol{A}\boldsymbol{\vartheta} = \boldsymbol{0}$, and $c > 0$ is a penalty parameter. The quadratic term $\|\boldsymbol{A}\boldsymbol{\vartheta}\|^2$ is a coupling term linking the local variables $\boldsymbol{\vartheta} = (\theta_1, ..., \theta_n)^\top$.

The proximal gradient primal-dual algorithm (Prox-GPDA) [15] considers using a primal step which minimizes a linearized version of (12), and the dual step which performs gradient ascent:

$$\boldsymbol{\vartheta}^{t+1} \leftarrow \operatorname*{arg\,min}_{\boldsymbol{\vartheta} \in \mathbb{R}^n} \Big\{ \Big\langle \underbrace{\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) + \boldsymbol{A}^\top \boldsymbol{\mu}^t + c\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\vartheta}^t}_{= \nabla_{\boldsymbol{\vartheta}} \mathcal{L}(\boldsymbol{\vartheta}^t, \boldsymbol{\mu}^t)}, \boldsymbol{\vartheta} - \boldsymbol{\vartheta}^t \Big\rangle + \frac{1}{2}\|\boldsymbol{\vartheta} - \boldsymbol{\vartheta}^t\|^2_{\boldsymbol{\Upsilon} + 2c\boldsymbol{D}} \Big\}, \tag{13}$$

$$\boldsymbol{\mu}^{t+1} \leftarrow \boldsymbol{\mu}^t + c \underbrace{\boldsymbol{A}\boldsymbol{\vartheta}^{t+1}}_{= \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\vartheta}^{t+1}, \boldsymbol{\mu}^t)}, \tag{14}$$

where $t = 0, 1, ...$ is the iteration number, $\nabla \boldsymbol{f}(\boldsymbol{\vartheta}) := \frac{1}{n}(\nabla f_1(\theta_1), \dots, \nabla f_n(\theta_n))^\top$ stacks up the gradients, and $\boldsymbol{\Upsilon} := \operatorname{diag}(\beta_1, \dots, \beta_n)$ is a diagonal matrix. Eq. (13), (14) lead to:

$$\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) + \boldsymbol{A}^\top \boldsymbol{\mu}^t + (c\boldsymbol{A}^\top \boldsymbol{A})\boldsymbol{\vartheta}^t + (\boldsymbol{\Upsilon} + 2c\boldsymbol{D})(\boldsymbol{\vartheta}^{t+1} - \boldsymbol{\vartheta}^t) = \boldsymbol{0},\ \boldsymbol{A}^\top \boldsymbol{\mu}^{t+1} = \boldsymbol{A}^\top \boldsymbol{\mu}^t + c\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\vartheta}^{t+1}, \tag{15}$$

respectively. Setting $p_i^t := \sum_{j|(i,j)\in\mathcal{E}} \mu_j^t$ shows that (15) can be decomposed into n parallel updates:

$$\textsf{Prox-GPDA:} \quad \begin{array}{rcl} \theta_i^{t+1} & \leftarrow & \frac{1}{\beta_i + 2cd_i}\Big\{\beta_i\theta_i^t - \nabla f_i(\theta_i^t) - p_i^t + c\sum_{j|(i,j)\in\mathcal{E}}(\theta_i^t + \theta_j^t)\Big\}, \quad i = 1, \dots, n, \\ p_i^{t+1} & \leftarrow & p_i^t + c\sum_{j|(i,j)\in\mathcal{E}}(\theta_i^{t+1} - \theta_j^{t+1}), \quad i = 1, \dots, n. \end{array} \tag{16}$$

This is a distributed algorithm implementable using *one* message exchange (i.e., getting $\sum_{j|(i,j)\in\mathcal{E}} \theta_j^{t+1}$) and *one* DO evaluation per iteration. By lending to the proofs for general primal-dual algorithms, [15] shows that, under proper c, $\{\beta_i\}$, it requires at most $\mathcal{O}(1/\epsilon)$ iterations to find an ϵ-stationary solution.

Eq. (16) shows that a distributed algorithm with balanced *computation* and *communication* cost can be derived from the primal-dual method. Furthermore, the method has a strong connection with existing distributed algorithms. First, we note that the inexact consensus ADMM (IC-ADMM) [16, Algorithm 2] which applies ADMM with inexact gradient update follows exactly the same form as (16). To see the connection for other algorithms, let us subtract Eq. (15) at the tth iteration by the $(t-1)$th one:

$$\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t-1}) + \boldsymbol{A}^\top(\boldsymbol{\mu}^t - \boldsymbol{\mu}^{t-1}) + c\boldsymbol{A}^\top \boldsymbol{A}(\boldsymbol{\vartheta}^t - \boldsymbol{\vartheta}^{t-1}) + (\boldsymbol{\Upsilon} + 2c\boldsymbol{D})(\boldsymbol{\vartheta}^{t+1} - 2\boldsymbol{\vartheta}^t + \boldsymbol{\vartheta}^{t-1}) = \boldsymbol{0}.$$

As $\boldsymbol{A}^\top(\boldsymbol{\mu}^t - \boldsymbol{\mu}^{t-1}) = c\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\vartheta}^t$, we have an equivalent form of Prox-GPDA algorithm:

$$\boldsymbol{\vartheta}^{t+1} = \big(\boldsymbol{I}_n - c(\boldsymbol{\Upsilon} + 2c\boldsymbol{D})^{-1}\boldsymbol{A}^\top \boldsymbol{A}\big)(2\boldsymbol{\vartheta}^t - \boldsymbol{\vartheta}^{t-1}) - (\boldsymbol{\Upsilon} + 2c\boldsymbol{D})^{-1}(\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t-1})). \tag{17}$$

We see that Prox-GPDA has various equivalent forms in (13) – (17). Below we show that a number of existing algorithms share similar communication-computation steps as Prox-GPDA.

a) Decentralized Gradient Descent (DGD): Initially proposed by [13] for convex problems, the DGD algorithm is one of the most popular distributed algorithms. This algorithm makes use of the penalized problem $\min_{\boldsymbol{\vartheta}\in\mathbb{R}^n} \sum_{i=1}^n f_i(\theta_i) + \frac{1}{2\alpha}\|\boldsymbol{\vartheta}\|^2_{\boldsymbol{I}_n - \boldsymbol{W}}$, where $\alpha > 0$ is a penalty parameter, and applies the gradient descent method with step size α to yield

$$\mathsf{DGD}: \quad \boldsymbol{\vartheta}^{t+1} \leftarrow \boldsymbol{W}\boldsymbol{\vartheta}^t - \alpha\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t), \quad \forall\, t = 0, 1\ldots.. \tag{18}$$

The update formula of DGD can be derived from Prox-GPDA as we consider (15) with $\boldsymbol{\mu}^t = \boldsymbol{0}\ \forall t$, $\boldsymbol{\Upsilon} + 2c\boldsymbol{D} = \alpha^{-1}\boldsymbol{I}_n$, and $\boldsymbol{W} = \boldsymbol{I}_n - c\alpha\boldsymbol{A}^\top\boldsymbol{A}$. However, unlike Prox-GDPA, to guarantee convergence to a stationary solution, DGD requires a diminishing step size as $\alpha^t = 1/t$ and an *additional assumption* that $\nabla f_i(\theta_i)$ is bounded for any θ_i and $i = 1, ..., n$ [17, Theorem 2].

b) EXTRA: The EXTRA algorithm was proposed in [18] as an alternative to DGD with convergence guarantee using a constant step size. Again, using the mixing matrix $\boldsymbol{W}$, the algorithm is described as follows. First, we initialize by $\boldsymbol{\vartheta}^1 \leftarrow \boldsymbol{W}\boldsymbol{\vartheta}^0 - \alpha\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^0)$, then

$$\mathsf{EXTRA}: \quad \boldsymbol{\vartheta}^{t+1} \leftarrow (\boldsymbol{I}_n + \boldsymbol{W})\boldsymbol{\vartheta}^t - \frac{1}{2}(\boldsymbol{I}_n + \boldsymbol{W})\boldsymbol{\vartheta}^{t-1} - \alpha[\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t-1})],\ \forall\, t = 1, 2, ... \tag{19}$$

A distinctive feature of (19) is that it computes the weighted difference between the previous two iterates $\boldsymbol{\vartheta}^t$ and $\boldsymbol{\vartheta}^{t-1}$. Interestingly, the above form of EXTRA is a special instance of Prox-GPDA. Setting $\boldsymbol{\Upsilon} + 2c\boldsymbol{D} = \alpha^{-1}\boldsymbol{I}_n$ in (17), we obtain

$$\boldsymbol{\vartheta}^{t+1} \leftarrow (2\boldsymbol{I}_n - 2c\alpha\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{\vartheta}^t - \frac{1}{2}(2\boldsymbol{I}_n - 2c\alpha\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{\vartheta}^{t-1} - \alpha[\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t-1})]. \tag{20}$$

Choosing $\boldsymbol{W} = \boldsymbol{I}_n - 2c\alpha \boldsymbol{A}^T\boldsymbol{A}$ and the above update recovers (19). The original proof in [18] assumes convexity for (1), but due to the above stated equivalence, the proof for Prox–GPDA for non-convex problems carries over to the EXTRA algorithm. It is worth mentioning that the original EXTRA algorithm in [18] takes a slightly more general form. That is, the term $\frac{1}{2}(\boldsymbol{I}_n + \boldsymbol{W})$ in (19) can be replaced by another mixing matrix $\tilde{\boldsymbol{W}}$, which satisfies $\frac{\boldsymbol{I}_n+\boldsymbol{W}}{2} \succeq \tilde{\boldsymbol{W}} \succeq \boldsymbol{W}$, and $\mathrm{null}\{\boldsymbol{I}_n - \tilde{\boldsymbol{W}}\} = \mathrm{span}\{\boldsymbol{1}\}$. However, it is not clear if this general form works for the non-convex distributed problems.

c) Rate Optimal Schemes: A fundamental question about distributed problem (1) is: "*what are the minimum computation and communication cost required to find an ϵ-stationary solution?*" An answer to this question is given in [19]. For any distributed algorithm using gradient information, it requires at least $\Omega\big((\epsilon\sqrt{\xi(\boldsymbol{L}_G)})^{-1}\big)$ communication rounds, and $\Omega\big(\epsilon^{-1}\big)$ rounds of gradient evaluation[1] to attain an ϵ-stationary solution, where $\xi(\boldsymbol{L}_G) := \frac{\lambda_{\min}(\boldsymbol{L}_G)}{\lambda_{\max}(\boldsymbol{L}_G)}$ is the ratio between the smallest non-zero and the largest eigenvalues of the graph Laplacian matrix $\boldsymbol{L}_G$. Interestingly, Prox-GPDA, EXTRA and IC-ADMM achieve the lower communication and computation bounds in star or fully connected networks. For general network topology, [19] proposed a *near optimal scheme* called xFILTER, which updates $\boldsymbol{\vartheta}$ by considering:

$$\boldsymbol{\vartheta}^{t+1} = \arg\min_{\boldsymbol{\vartheta}\in\mathbb{R}^n} \left\{ \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t)^\top(\boldsymbol{\vartheta} - \boldsymbol{\vartheta}^t) + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\vartheta} + \frac{c}{2}\|\boldsymbol{A}\boldsymbol{\vartheta}\|^2 + \frac{1}{2}\|\boldsymbol{\vartheta} - \boldsymbol{\vartheta}^t\|^2_{\boldsymbol{\Upsilon}} \right\}. \tag{21}$$

Compared to (13), the quadratic term $\frac{c}{2}\|\boldsymbol{A}\boldsymbol{\vartheta}\|^2$ is not linearized. This term couples the local variables, so (21) itself does not lead to a distributed update for $\boldsymbol{\vartheta}^{t+1}$. To resolve this issue, the authors proposed to generate $\boldsymbol{\vartheta}^{t+1}$ by using the Qth order Chebychev polynomial to approximately solve (21). They showed that setting $Q = \widetilde{\mathcal{O}}\big(1/\sqrt{\xi(\boldsymbol{L}_G)}\big)$ suffices to produce an algorithm that requires $\widetilde{\mathcal{O}}\big((\epsilon\sqrt{\xi(\boldsymbol{L}_G)})^{-1}\big)$ communication rounds and $\mathcal{O}(\epsilon^{-1})$ gradient evaluations rounds, where the notation $\widetilde{\mathcal{O}}(\cdot)$ hides a $\log$ function of n (which is usually small). This matches the aforementioned lower bounds. Similarly, the communication effort required is near-optimal (up to a multiplicative logarithmic factor). Further, by comparing with all other batch methods (to be) reviewed in this work, this is the only algorithm whose gradient evaluation complexity is *independent* of the graph structure.

Gradient-Tracking Based Methods. Another class of algorithms that can deal with the non-convex problem (1) leverages the technique of *gradient tracking* (GT). The method is based on the simple idea that, if every agent has access to the global gradient $1/n\sum_{i=1}^n \nabla f_i(1/n\sum_{j=1}^n \theta_j^t)$, then the (centralized) GD can be performed at each agent. The GT technique provides an iterative approach to do so approximately.

[1]In each gradient evaluation, each local node i evaluates $\nabla f_i(\cdot)$ once.

The algorithm performs two message exchanges each iteration [with $\hat{\boldsymbol{W}}$ satisfying (10)]:

$$\mathsf{GT}: \quad \boldsymbol{\vartheta}^{t+1} \leftarrow \hat{\boldsymbol{W}}\boldsymbol{\vartheta}^t - \alpha \boldsymbol{g}^t, \;\; \boldsymbol{g}^{t+1} \leftarrow \hat{\boldsymbol{W}}\boldsymbol{g}^t + \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t+1}) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t), \; \forall\; t = 1, 2, \ldots. \tag{22}$$

The ith element g_i^t of $\boldsymbol{g}^t$ is the local estimate of the global gradient at each agent i, obtained by mixing the estimates of its neighbors and refreshing its local ∇f_i. As shown in [20], the GT algorithm (22) converges at a rate of $\mathcal{O}(1/\epsilon)$ to a stationary point. One key strength of the GT based methods is that they can also work in directed and time-varying graphs; see [20] for more discussions.

We remark that the GT based method is related to the general form of EXTRA. To see this, we subtract the updates of $\boldsymbol{\vartheta}^{t+1}$ and $\boldsymbol{\vartheta}^t$, and apply the update of $\boldsymbol{g}^t$ to obtain

$$\boldsymbol{\vartheta}^{t+1} \leftarrow 2\hat{\boldsymbol{W}}\boldsymbol{\vartheta}^t - \hat{\boldsymbol{W}}^2\boldsymbol{\vartheta}^{t-1} - \alpha\left(\nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t-1})\right), \; \forall\; t = 1, 2, ... \tag{23}$$

It can be shown that if $\hat{\boldsymbol{W}}$ satisfies $\hat{\boldsymbol{W}} \succeq (\hat{\boldsymbol{W}})^2 \succeq 2\hat{\boldsymbol{W}} - \boldsymbol{I}_n$, then the algorithm takes the same form as the generalized EXTRA discussed after (20); see [21, Section 2.2.1]. However, the analysis for generalized EXTRA only works on convex problems, and does not carry to the GT method in the non-convex setting.

We remark that all the above algorithms converge for the challenging Example 3 except for the DGD algorithm. This is because for the latter example, the gradient can be unbounded.

B. Algorithms for Streaming Data

In the streaming data setting, the data oracle returns $\mathsf{DO}_i(\boldsymbol{\theta}_i^t)$ which is an unbiased estimator of $\nabla f_i(\boldsymbol{\theta}_i^t)$ with finite variance under Assumption 2. This data model is typical in processing large-to-infinite datasets. In this setting, balancing between *communication* and *computation* cost is an important issue since even the centralized algorithm may have slow convergence. The first study of distributed stochastic algorithm dates back to Tsitsiklis et al. [22] which studied the asymptotic convergence of the DSGD algorithm reviewed below. The DSGD algorithm is relevant to the distributed estimation problem important in adaptive signal processing, as such, many works are devoted to studying its transient behavior (of bias, mean-squared error, etc.), e.g., [23], [24] and the overview in [5]. Unfortunately, these works are mainly focused on *convex problems*. Below, we review the more recent results dedicated to the non-convex learning setting with non-asymptotic convergence analysis.

a) Distributed Stochastic Gradient Descent (DSGD): This class of algorithm replaces the deterministic oracle in the DGD with the stochastic oracle (5). It takes the following form:

$$\mathsf{DSGD}: \quad \boldsymbol{\vartheta}^{t+1} \leftarrow \boldsymbol{W}\boldsymbol{\vartheta}^t - \alpha^t \mathsf{DO}(\boldsymbol{\vartheta}^t), \tag{24}$$

where $\alpha^t > 0$ is the step size and we defined $\mathsf{DO}(\boldsymbol{\vartheta}^t) := (\mathsf{DO}_1(\theta_1^t), ..., \mathsf{DO}_n(\theta_n^t))^\top$. Obviously, DSGD can be implemented in a distributed manner via the mixing matrix $\boldsymbol{W}$. The study of such an algorithm in the non-convex setting dates back to the work [22]. Among other results, the authors showed that if the step size sequence satisfies $\alpha^t \le c/t$ for some $c > 0$, the DSGD algorithm converges almost surely to a first-order stationary solution. However, [22] mainly provides asymptotic convergence conditions, without a clear indication of whether DSGD can outperform its centralized counterpart.

Recently, DSGD [a.k.a. decentralized parallel stochastic gradient descent (D-PSGD)] has been applied for decentralized training of neural networks in [3], and the convergence rate has been analyzed in [25]. In the analysis by [25], the following condition on the data across agents is assumed:

$$n^{-1} \textstyle\sum_{i=1}^n |\nabla f_i(\theta) - \nabla f(\theta)|^2 \le \varsigma^2 < \infty, \ \forall \ \theta \in \mathbb{R}. \tag{25}$$

Such an assumption can be difficult to verify, and it is *only* required when analyzing the convergence rate of DSGD for non-convex problems. For example, if the loss function is quadratic, then the corresponding gradient is a linear function of θ, e.g., $\nabla f_i(\theta) = a_i\theta + b_i$, $i = 1, ..., n$. The LHS of (25) is unbounded if $a_i \ne (1/n)\sum_{j=1}^n a_j$, *i.e.,* whenever the cost function is heterogeneous.

Under (25) and Assumption 1 & 2, for any *sufficiently large* T, if we set $\alpha^t = \mathcal{O}(\sqrt{n/(\sigma^2 T)})$ for all $t \ge 0$, then the DSGD finds an approximate stationary solution to (1) satisfying $\mathbb{E}[\mathsf{Gap}(\boldsymbol{\vartheta}^{\tilde{t}})] = \mathcal{O}(\sigma/\sqrt{nT})$, where $\tilde{t}$ is uniformly drawn from $\{1, ..., T\}$ [25, Corollary 2]. Compared to the centralized SGD algorithm, a speedup factor of $1/\sqrt{n}$ is observed, which is due to the variance reduction effect by averaging from n nodes. Yet achieving this requires $\varsigma^2 = \mathcal{O}(1)$ such that the data is *homogeneous* across the agents. Also, see [11], [12] which show that the DSGD algorithm converges to a second-order stationary solution under a similar condition as (25).

In summary, the DSGD algorithm is simple to implement, but it has a major limitation when dealing with heterogeneous data. Such a limitation will also be demonstrated in our numerical experiments.

b) D^2 Algorithm: To relax the local data assumption (25) from DSGD, an algorithm named D^2 has been proposed in [26]. Again, using the mixing matrix $\boldsymbol{W}$, the recursion of D^2 is given as:

$$\mathsf{D}^2: \quad \boldsymbol{\vartheta}^{t+1} \leftarrow 2\boldsymbol{W}\boldsymbol{\vartheta}^t - \boldsymbol{W}\boldsymbol{\vartheta}^{t-1} - \alpha^t \boldsymbol{W}\left(\mathsf{DO}(\boldsymbol{\vartheta}^{\mathsf{t}}) - \mathsf{DO}(\boldsymbol{\vartheta}^{\mathsf{t}-1})\right), \ \forall \ t \ge 0. \tag{26}$$

In addition to the previous conditions on the weight matrix (10), D^2 requires a special condition $\lambda_{\min}(\boldsymbol{W}) > -1/3$. Basically, the condition implies that the weight of combining the current node is greater than the ones of combining its neighbors. Together with Assumption 1 & 2, for any sufficiently large T, we set

$\alpha^t = \mathcal{O}(\sqrt{n/(\sigma^2 T)})$ for all $t \geq 0$, and D^2 finds an approximate stationary solution [26] to (1) satisfying $\mathbb{E}[\|n^{-1}\sum_{j=1}^{n} \nabla f_j(\overline{\boldsymbol{\theta}}^{\tilde{t}})\|^2] = \mathcal{O}(\sigma/\sqrt{nT})$, where $\tilde{t}$ is uniformly drawn from $\{1, ..., T\}$.

In fact, comparing (26) to (23) reveals a close similarity between D^2 and GT: both algorithms use the current and the previous DO's, and both require *two* local communication rounds per iteration. The difference is that the GT method applies a squared mixing matrix $\boldsymbol{W}^2$ on $\boldsymbol{\vartheta}^{t-1}$ instead of the mixing matrix $\boldsymbol{W}$ for D^2, and there is a $\boldsymbol{W}$ multiplying the difference of the gradient estimates. Such a seemingly minor difference turns out to be one major limiting factor for D^2; see the example below.

Example 4. *[27] Consider a line network consisting of three nodes, with $f_i(x) = (x - b_i)^2,\ i = 1, 2, 3$ (for some fixed b_i), and mixing matrix: $\boldsymbol{W} = [0.5, 0.5, 0; 0.5, 0, 0.5; 0, 0.5, 0.5]$, which has eigenvalues $\{-0.5, 0.5, 1\}$. One can show D^2 diverges for any constant $\alpha^t \leq 0.25$, or diminishing step size $\alpha^t = 1/t$.*

c) Distributed Stochastic Gradient Tracking: How to design algorithms that can deal with heterogeneous data, while requiring conditions weaker than that of D^2? An algorithm called Gradient-tracking based Non-convex Stochastic algorithm for Decentralized training (GNSD) has been proposed in [28], which is essentially a stochastic version of the GT method in (23):

$$\mathsf{GNSD}: \ \boldsymbol{\vartheta}^{t+1} \leftarrow 2\boldsymbol{W}\boldsymbol{\vartheta}^t - \boldsymbol{W}^2\boldsymbol{\vartheta}^{t-1} - \alpha^t\left(\mathsf{DO}(\boldsymbol{\vartheta}^{\mathsf{t}}) - \mathsf{DO}(\boldsymbol{\vartheta}^{\mathsf{t}-1})\right). \tag{27}$$

It is shown that GNSD has the similar convergence guarantees as D^2, without requiring the assumption (25) and the condition $\lambda_{\min}(\boldsymbol{W}) > -1/3$.

To summarize, D^2 and GNSD address the challenge of heterogeneous data unique to the streaming data setting, while simple methods such as DSGD require data to be homogeneous. On the other hand, D^2 and GNSD require additional communication per iteration compared with DSGD. We remark that there appears to be no work extending primal-dual type algorithm/analysis to the streaming setting.

C. Other Distributed Algorithms

Despite the differences in DOs used and assumptions needed for convergence, the reviewed algorithms may be regarded as variants of unconstrained gradient descent methods for a single parameter (vector) on a fixed communication graph. However, special communication and computation architectures may arise in practice. Here we conclude the section by highlighting a few works in relevant directions.

a) Coordinate Descent Methods: When the optimization model (1) involves multiple variables, it is often beneficial to adopt a *coordinate descent* method, which optimizes only one variable at a time, holding the others as constant. An example is *matrix factorization* problem discussed in Example 2. In specific, [15],

[29] respectively proposes to combine Prox-GPDA, GT with coordinate descent to tackle the distributed dictionary learning problem (*batch* data), with convergence guarantee.

b) Directed and Time Varying Graphs: Throughout this paper, we have assumed that the graph connecting the agents is undirected and static. However, directed and/or time-varying graph topology may arise in practice, e.g., with unreliable network. Several works have been proposed for various settings. For *batch data*, [20] proposed the SONATA algorithm which combines GT with PushSum technique; for *streaming data*, [30] proposed the Stochastic Gradient Push (SGP) algorithm which combines SGD and PushSum technique. Both SONATA and SGP are shown to converge sublinearly to a stationary solution on time-varying and directed graphs.

IV. Practical Issues and Numerical Results

We discuss the practical issues related to the implementation of distributed algorithms. We aim to demonstrate how system and algorithm parameters, such as network size, computation/communication speed, batch size and model size, should be considered jointly to decide on the most suitable algorithm. In particular, we compare their effects on the overall runtime performance of algorithms.

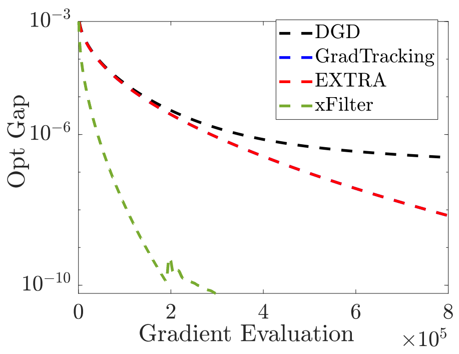
Our experiments are conducted on two different computer clusters, one provided by Minnesota Supercomputing Institute (MSI), another by Amazon Web Services (AWS). The MSI cluster has better independent computation power at each node, but worse communication bandwidth than the AWS cluster; see Fig. 3 (right). Specifically, MSI nodes have Intel Haswell E5-2680v3 CPUs at 3.2 GHz, 14Gbps inter-node communication, while AWS nodes have Intel Xeon E5-2686v4 CPUs at 3.0 GHz, NVIDIA K80 GPUs and 25Gbps inter-node communication.

Two sets of experiments are conducted. The first set compares different algorithms on a single machine. Since the distributed implementation is only simulated, the purpose of this set is to understand the theoretical behavior of algorithms. The second set of experiments showcases the algorithm performance on truly distributed systems. These algorithms are implemented in Python 3.6 with the MPI communication protocol. We benchmark the algorithms by using the gap $\mathsf{Gap}(\boldsymbol{\vartheta})$ in (2).

a) Experiment Set I: We consider tackling a regularized logistic regression problem with a *non-convex* regularizer in a distributed manner. We use similar notations as in Example 1, *i.e.,* the feature is $\boldsymbol{x}_i^\ell$ and the label is y_i^ℓ. Let $\lambda, \rho > 0$ be the regularizer's parameters, each local cost function f_i is given by

$$f_i(\boldsymbol{\theta}_i) = \frac{1}{M_i} \sum_{\ell=1}^{M_i} \log\left(1 + \exp(-y_i^\ell \boldsymbol{\theta}_i^\top \boldsymbol{x}_i^\ell)\right) + \lambda \sum_{s=1}^{d} \frac{\rho\theta_{i,s}^2}{1+\rho\theta_{i,s}^2}.$$

All algorithms are implemented in MATLAB. We set the dimension at $d = 10$, and generate $M_i = 400$ synthetic data points on each of the $n = 32$ agents; the communication network is a random regular graph



Cost per iter.	**Computation (m)**				**Communication (n)**		
Settings	128	8	64	256	2	8	32
MSI, DSGD	1				0.30	2.38	8.87
MSI, GNSD	1				0.64	4.78	19.4
AWS, DSGD	1				0.14	1.47	4.12
AWS, GNSD	1				0.17	1.60	4.21
MSI, DSGD		1	1.09	1.36			2.61
MSI, GNSD		1	1.12	1.45			8.47

Fig. 3: (Left) Stationarity gap against iteration number of different algorithms with a synthetic dataset and $n = 32$ agents. Note that the curve for Gradient Tracking and EXTRA overlaps with each other. (Right) Normalized running time *per iteration / message exchange round* on the MSI and AWS clusters under different settings for batch size m and network size n.

of degree 5. The stationarity gap versus iteration number for the surveyed batch algorithms is shown in Fig. 3 (left). As seen, in terms of total number of full gradient evaluations, the xFILTER is the fastest. The observation we made is also consistent with the theoretical prediction, because the discussion in Sec. III-A suggests that xFILTER is the only algorithm whose total gradient evaluation is *independent* of the graph structure, and matches the centralized GD.

b) Experiment Set II: We focus on the DSGD and GNSD algorithms for streaming data, and apply them to train a neural network as in Example 1, and the task to classify handwritten digits from the MNIST dataset. The neural network contains two hidden layers with 512 and 128 neurons each, and 4.68×10^5 parameters in total. The training data set has 4.8×10^4 entries and is divided evenly among n nodes. The DSGD and GNSD algorithms adopt the streaming data oracle in Sec. II), and all agents use the same mini-batch sizes $m_t = m$. The communication graph is a random regular graph with degree 5.

Before we compare the overall performance of different algorithms, we first examine the computation/communication performance for our two clusters in running DSGD/GSND. In the upper part of Fig. 3 (right), we compare the *relative* computation and communication costs on MSI and AWS. It is clear that the AWS cluster has better communication efficiency compared to the MSI. For example, consider running GNSD on a network with $n = 8$ nodes, and set the computational time per iteration as 1 unit of time. Observe that AWS uses 1.6 units of time on communication, while MSI uses 4.78 units.

Network Scalability. We analyze how the network size n affects the overall convergence speed. Intuitively, if the communication cost is relatively cheaper than that of computation, then it is beneficial to use a larger network and involve more agents to share the computational burden. In Fig. 4 (left) & (middle), we see that the runtime performance of DSGD/GNSD algorithms on AWS significantly improves as

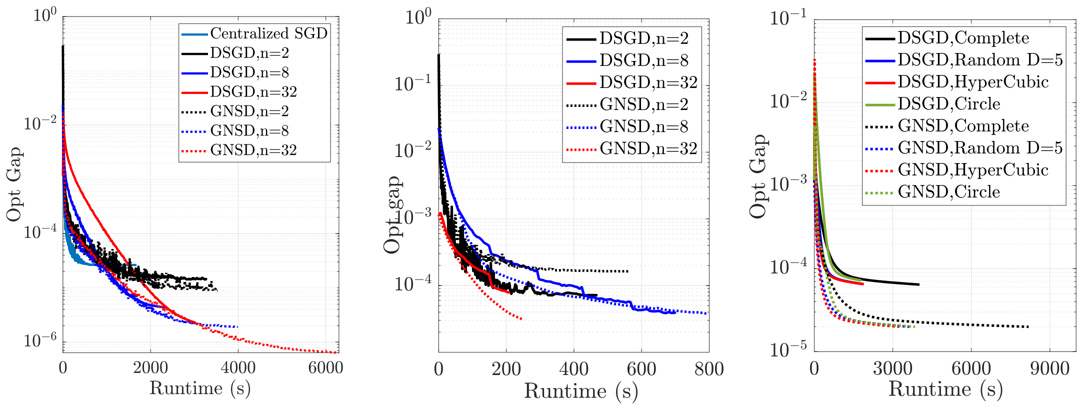


Fig. 4: Runtime comparison of streaming algorithms: (Left) on MSI with $n = 1, 2, 8, 32$ agents, batchsize $m = 128$ for all algorithms, terminated in 450 epochs; (Middle) on AWS with $n = 2, 8, 32$ agents, batchsize $m = 128$, terminated in 128 epochs; (Right) on MSI with different types of graph topologies with $n = 32$ agents, batchsize $m = 128$, terminated in 256 epochs.

the number of nodes increases (from $n = 8$ to $n = 32$); while there is no significant improvement for the experiments on MSI. This confirms our intuition since AWS has a high speed communication network. Besides, one can observe in the left figure the benefit of distributed learning ($n > 1$) over the centralized scheme ($n = 1$), where DSGD with multiple agents can reach a smaller optimality gap. On both platforms, we observe that GNSD achieves even smaller optimality gap compared with DSGD, but requires more time to complete the given number of epochs. This is reasonable since as discussed in Sec. III-B, DSGD requires one round of communication per evaluation of DO, while GNSD requires two.

Graph Topology. Another key parameter with a significant impact on the algorithm performance is the graph topology. It is important to note that, although theoretical analysis indicates that well-connected graphs [which have large $\xi(\boldsymbol{L}_G)$] has a faster convergence rate, in practice factors such as the maximum degree of agents also matter. In Fig. 4 (right), we compare the runtime with $n = 32$ agents on different types of topology – including a complete graph, a random regular graph with degree 5, a hypercube graph, and a circle graph. We observe that well-connected *sparse* graphs (e.g., random regular, hybercube) are preferred, since there are less communication overheads compared with dense graphs (e.g., complete graph) and poorly-connected graphs (e.g., circle graph).

Mini-Batch Size. The choice of mini-batch size m is another important parameter. While it speeds up the convergence with a large mini-batch size, it can be computationally expensive and requires extensive memory. We examine the tradeoff with the mini-batch size in Fig. 5 (left), where the experiments are run on the MSI cluster. As seen, increasing the batch size improves the GNSD algorithm more significantly than DSGD. Further, in the lower part of Fig. 3 (right), we provide the normalized per-

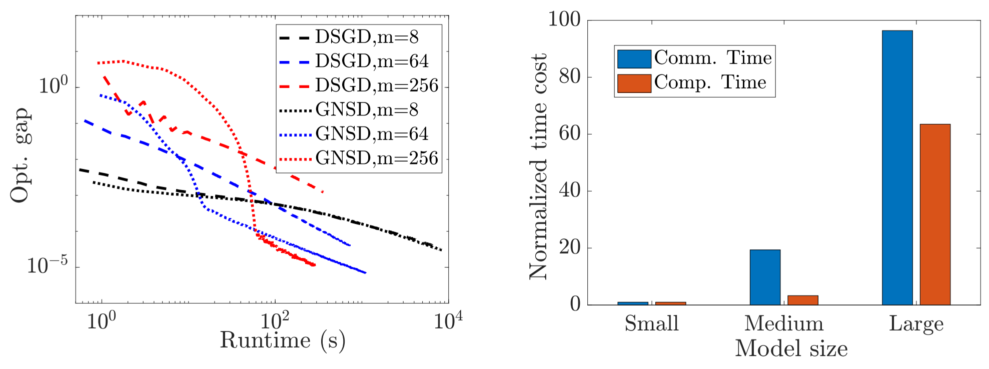


Fig. 5: (Left) Runtime comparison of mini-batch size $m = 8, 64, 256$ on MSI, terminated after 256 epochs. Data is heterogeneous, where each node is assigned exclusive classes; (Right) Normalized computation and communication cost (normalized to the small model) for different model sizes on MSI with $n = 32$ agents.

iteration computation and communication time with different mini-batch sizes. Notice that for DSGD, it takes 1.09 and 1.36 times of *computation* time with a mini-batch size of $m = 64$ and $m = 256$, compared to the baseline setting with $m = 8$. A larger mini-batch size seems to be more efficient.

Heterogeneous Data. We illustrate the effect of heterogeneous data on different algorithms by again using Fig. 5. In this experiment, we divide the data according to their labels, and assign each agent exclusively with two classes. We can see that the performance of DSGD becomes significantly worse compared with GNSD, especially when the batch size becomes larger (in which case the variance caused by sampling becomes smaller, hence the effect of heterogeneous data is more pronounced). This observation corroborates the theoretical results in Sec. III-B, that GNSD does not require any assumption on the distribution of the data, while DSGD does.

Model Size. Intuitively, small model may benefit from distributed algorithm due to the small amount of information exchange required, especially on systems where communication is slower than computation. As shown in Fig 5 (right), we compare 3 neural networks – a small network (2-layer fully connected neural network, with 8×10^3 parameters), a medium network (LeNet-5 with 2 convolutional layers and 3 fully connected layers, with 6×10^4 parameters), a large network (Keras example for MNIST with 4 convolutional layers and 3 fully connected layers, with 4.07×10^5 parameters), run on the MSI cluster with DSGD. As model size increases, the growth of communication cost outweighs the computation cost.

c) Other Related Issues: Another active research direction is on improving communication efficiency in distributed algorithms. Taking the DSGD as an example, a possible idea is to perform SGD updates locally for multiple times (say I) at an agent before exchanging the parameters with neighbors. Using this scheme, [31] shows that with $I = \Theta(1/\epsilon)$, the distributed algorithm run on a *star graph* topology requires only $\mathcal{O}(1/\epsilon)$ [*resp.* $\mathcal{O}(1/\epsilon^{\frac{3}{2}})$] message exchanges for homogeneous (*resp.* heterogeneous) dataset to find an ϵ-stationary solution to (1). Alternatively, [32] proposes to skip unnecessary communication steps when the deviation of local variables is small. Lastly, to reduce the time cost on synchronizing over agents and to make distributed learning less vulnerable to straggling agents, there are works that allow for asynchronous communication; see [30], [33] for example.

V. Conclusions & Open Problems

This paper provides a selected review on recent developments of non-convex, distributed learning algorithms. We show the interplay between *problem, data* and *computation, communication*, leading to different algorithms. We also compare the algorithms using numerical experiments on computer clusters, showing their practical potentials. Below we list a few directions for future research.

a) Dynamical Data: Beyond batch and streaming data, an open problem is to develop distributed algorithms for *dynamical* data. We consider a DO which takes the same form as the first equation of (5), but the data samples $\{\xi_{i,\ell}^{t+1}\}_{\ell=1}^{M_\ell}$ are drawn instead from a *parameterized* distribution $\pi_i(\cdot;\boldsymbol{\vartheta}^t)$. The new data model corresponds to a *dynamic* data acquisition process controlled by the iterates. The output of this DO will be used by the algorithm to compute the next iterate. For example, this is relevant to policy optimization where $\boldsymbol{\vartheta}^t$ is the joint policy exercised by the agents, and the data acquired are state/action pairs generated through interactions with the environment (therefore dependent on the current policy $\boldsymbol{\vartheta}^t$); the state/action pairs will then be used to compute the policy gradient for updating $\boldsymbol{\vartheta}^{t+1}$.

Distributed algorithms based on the dynamic DO is challenging to analyze as computation, communication, and data acquisition have to be jointly considered. To the best of our knowledge, such setting has only been recently studied for a centralized algorithm in [34]. In a distributed setting, progresses have been made in multi-agent reinforcement learning, e.g., [35] applied a linear function approximation to simplify the non-convex learning problem as a convex one. Nevertheless, a truly distributed, non-convex algorithm with a dynamic DO has neither been proposed nor analyzed. Another challenging dynamic scenario is under the *online* setting, where no statistical assumption is imposed on the DO output. However, most of the developments are still restricted to convex problems; see [36].

b) Distributed Feature: In many applications, leveraging additional features from another domain or party can further improve the inference performance. However, data with these features may be private records and cannot be shared. This imposes a challenging question of how to enable the agents that own different sets of features to collaborate on the learning task; see [16], [37].

c) Federated & Robust Learning: To improve user privacy, federated learning (FL) [38] is proposed for distributed learning in edge networks. Unlike traditional distributed learning, FL emphasizes on the ability to deal with unbalanced data and poorly connected users. Security is another concern for FL and algorithms that are resilient to adversary attacks or model poisoning are crucial [39], [40] for example.

References

[1] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[2] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "GossipGraD: Scalable deep learning using gossip communication based asynchronous gradient descent," preprint, available at arXiv:1803.05880.

[3] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *Advances in Neural Information Processing Systems*, 2017.

[4] A. Nedic and A. Ozdaglar, "Cooperative distributed multi-agent optimization," in *Convex Optimization in Signal Processing and Communications*. Cambridge University Press, 2010.

[5] A. H. Sayed, S.-Y. Tu, J. Chen, X. Zhao, and Z. J. Towfic, "Diffusion strategies for adaptation and learning over networks: an examination of distributed strategies and network behavior," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 155–171, 2013.

[6] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 32–43, 2014.

[7] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, no. 2, pp. 117–129, Jun 1987. [Online]. Available: http://dx.doi.org/10.1007/BF02592948

[8] M. Hong, M. Razaviyayn, and J. Lee, "Gradient primal-dual algorithm converges to second-order stationary solution for nonconvex distributed optimization over networks," in *ICML*, 2018, pp. 2014–2023.

[9] A. Daneshmand, G. Scutari, and V. Kungurtsev, "Second-order guarantees of gradient algorithms over networks," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 359–365.

[10] B. Swenson, S. Kar, H. V. Poor, and J. M. F. Moura, "Annealing for distributed global optimization," *arXiv Preprint*, 2019, arXiv:1903.07258.

[11] S. Vlaski and A. H. Sayed, "Distributed learning in non-convex environments–part i: Agreement at a linear rate," *arXiv preprint arXiv:1907.01848*, 2019.

[12] ——, "Distributed learning in non-convex environments–part ii: Polynomial escape from saddle-points," *arXiv preprint arXiv:1907.01849*, 2019.

[13] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[14] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM review*, vol. 46, no. 4, pp. 667–689, 2004.

[15] M. Hong, D. Hajinezhad, and M.-M. Zhao, "Prox-PDA: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over networks," in *ICML*, 2017.

[16] T.-H. Chang, M. Hong, and X. Wang, "Multi-agent distributed optimization via inexact consensus ADMM," *IEEE Transactions on Signal Processing*, vol. 63, no. 2, pp. 482–497, Jan 2015.

[17] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Transactions on Signal Processing*, vol. 66, no. 11, pp. 2834–2848, June 2018.

[18] W. Shi, Q. Ling, G. Wu, and W. Yin, “EXTRA: An exact first-order algorithm for decentralized consensus optimization,” *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2014.

[19] H. Sun and M. Hong, “Distributed non-convex first-order optimization and information processing: Lower complexity bounds and rate optimal algorithms,” *IEEE Transactions on Signal processing*, July 2019, accepted for publication.

[20] G. Scutari and Y. Sun, “Distributed nonconvex constrained optimization over time-varying digraphs,” *Mathematical Programming*, vol. 176, no. 1-2, pp. 497–544, 2019.

[21] A. Nedic, A. Olshevsky, and W. Shi, “Achieving geometric convergence for distributed optimization over time-varying graphs,” *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

[22] J. Tsitsiklis, D. P. Bertsekas, and M. Athans, “Distributed asynchronous deterministic and stochastic gradient optimization algorithms,” *IEEE Transactions on Automated Control*, vol. 31, pp. 803–812, 1986.

[23] F. S. Cattivelli and A. H. Sayed, “Diffusion LMS strategies for distributed estimation,” *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1035–1048, 2009.

[24] S. Kar, J. M. Moura, and K. Ramanan, “Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication,” *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 3575–3605, 2012.

[25] 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,” in *NeurIPS*, 2017, pp. 5330–5340.

[26] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, “D^2: Decentralized training over decentralized data,” in *Proc. of the 35th International Conference on Machine Learning*, 10–15 Jul. 2018, pp. 4848–4856.

[27] J. Zhang and K. You, “Decentralized stochastic gradient tracking for empirical risk minimization,” *arXiv preprint arXiv:1909.02712*, 2019.

[28] S. Lu, X. Zhang, H. Sun, and M. Hong, “GNSD: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization,” in *Proc. of IEEE Data Science Workshop (DSW)*, Jun. 2019, pp. 315–321.

[29] A. Daneshmand, Y. Sun, G. Scutari, F. Facchinei, and B. M. Sadler, “Decentralized dictionary learning over time-varying digraphs,” *Journal of Machine Learning Research*, 2019.

[30] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, “Stochastic gradient push for distributed deep learning,” in *Proc. of International Conference on Machine Learning*, 2019, pp. 344–353.

[31] H. Yu, R. Jin, and S. Yang, “On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization,” in *International Conference on Machine Learning*, 2019, pp. 7184–7193.

[32] R. Aragues, G. Shi, D. V. Dimarogonas, C. Sagues, and K. H. Johansson, “Distributed algebraic connectivity estimation for adaptive event-triggered consensus,” in *2012 American Control Conference (ACC)*, June 2012, pp. 32–37.

[33] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *Proc. of ICML*, July 10-15, 2018, pp. 3049–3058.

[34] B. Karimi, B. Miasojedow, E. Moulines, and H.-T. Wai, “Non-asymptotic analysis of biased stochastic approximation scheme,” in *Conference on Learning Theory*, 2019.

[35] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Başar, “Fully decentralized multi-agent reinforcement learning with networked agents,” in *International Conference on Machine Learning*, 2018, pp. 9340–9371.

[36] S. Shahrampour and A. Jadbabaie, “Distributed online optimization in dynamic environments using mirror descent,” *IEEE Transactions on Automatic Control*, vol. 63, no. 3, pp. 714–725, 2017.

[37] Y. Hu, D. Niu, J. Yang, and S. Zhou, “FDML: A collaborative machine learning framework for distributed features,” in *Proc. ACM International Conference on Knowledge Discovery & Data Mining*, Aug. 2018, pp. 2232–2240.

[38] J. Konecny, H. B. McMahan, and D. Ramage, “Federated optimization: Distributed optimization beyond the datacenter,” in *Proc. of Optimization for Machine Learning*, 2015, pp. 1–5.

[39] Z. Yang and W. U. Bajwa, “Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning,” *IEEE Transactions on Signal and Information Processing over Networks*, 2019.

[40] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, “Analyzing federated learning through an adversarial lens,” in *Proc. of Machine Learning Research*, 2019.