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

Distributed Learning in the Nonconvex World

From batch data to streaming and beyond

Digital Object Identifier 10.1109/MSP.2020.2970170
Date of current version: 28 April 2020

Distributed learning has become a critical enabler of the massively connected world that many people envision. 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 unique perspective of how these elements should work together in an effective and coherent manner. In particular, we selectively review recent techniques developed for optimizing nonconvex models (i.e., problem classes) that process batch and streaming data (data types) across networks in a distributed manner (communication and computation paradigm). We describe the intuitions and connections behind a core set of popular distributed algorithms, emphasizing how to balance computation and communication costs. Practical issues and future research directions will also be discussed.

Introduction

We live in a highly connected world, and it will become exponentially more so within 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 [41]. These devices collect a huge amount of real-time data, perform complex computational tasks, and provide vital services that significantly improve our lives and enrich our collective productivity.

In a massively connected world, the four key elements discussed previously (namely, problems, data, communication, and computation) enable scalable distributed processing and real-time intelligence. They are closely tied to each other, as illustrated in Figure 1. For example, without a meaningful machine learning (ML) problem, using massive computational resources to crunch large amounts of data rarely leads to any actionable intelligence. Similarly, despite their sophisticated design and helpful interpretation from neural sciences, modern neural networks may not be successful without highly efficient computation methods. The overarching goal of this selective review is to provide a fresh point of view that relates how these elements should work together in the most effective and coherent manner to realize scalable processing, real-time intelligence, and, ultimately, contribute to the

vision of a smart, highly connected world. Some key aspects to be taken into consideration are:

- *Nonconvexity*: For many emerging distributed applications, the problems that distributed nodes must solve will be highly complicated. For instance, in distributed ML, the nodes (e.g., mobile devices) jointly learn a model based on local data (e.g., the images on each device). To accurately represent the local data, the nodes are often required to use nonconvex loss functions, such as those that compose multiple nonlinear activation functions, through collaborative deep learning [1]–[3].
- *Data acquisition processes*: One of the main reasons behind the success of ML is the ability to process data at scale. This means that one can quickly process large volumes of information (i.e., deal with batch data) and, more importantly, it requires the ability to handle streaming data. There is an urgent need (and, hence, a growing body of research) to accommodate the massive amount of streaming data from online review platforms (e.g., Amazon), social networks (e.g., Facebook), and so forth.
- *Distributed processing*: The growing network size, increasing amount of distributed data, and requirements for real-time response often make traditional centralized processing unviable. For example, self-driving cars should be carefully coordinated when they meet at intersections, but since every such vehicle can generate up to 40 gigabytes of data (e.g., from lidar and cameras) per second—an amount that overwhelms the fastest cellular networks—it is impossible to pool all the information for real-time central control. This

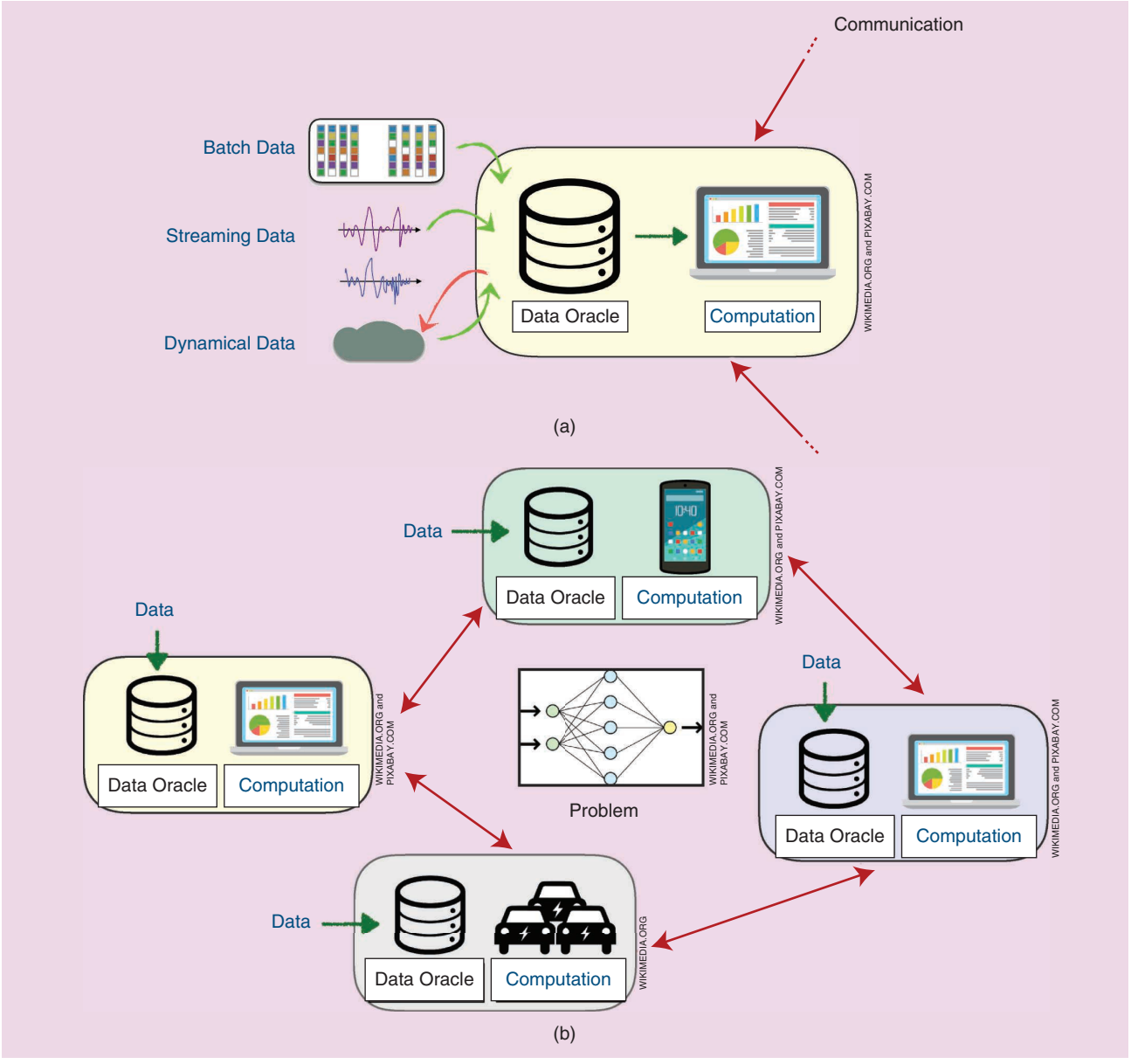


FIGURE 1. The overview of the key elements in distributed learning. (a) The flow between different elements at a single data agent (e.g., an image) is taken from diverse types through an oracle and processed locally before communicating with other agents. The goal is to tackle a nonconvex learning problem. (b) Distributed learning on a network of agents.

and other examples, from small and ordinary devices (e.g., in-home smart appliances) to large and vitally important infrastructure (e.g., national power-distribution networks), show how essential fast distributed processing will be to our collective well-being, productivity, and prosperity.

This article examines recent advances on distributed algorithms. Unlike existing articles [4]–[6], this one centers around nonconvex optimization and learning problems. We reveal connections and design insights related to a core set of first-order algorithms while highlighting the interplay among problems, data, computation, and communication. We hope that the connections we discuss will help the readers compare the theoretical and practical properties between algorithms and translate new features and theoretical advances developed for one kind of algorithm to equivalent types without the risk of "reinventing the wheel."

Problems and data models

Problem class

Consider n interconnected agents. The network connecting them 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 the agents is to find a solution, $\boldsymbol{\vartheta}^\star := (\boldsymbol{\theta}_1^\star, ..., \boldsymbol{\theta}_n^\star)$, that tackles the nonconvex optimization problem

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

where

$$\mathcal{H} := \{\boldsymbol{\theta}_i \in \mathbb{R}^d, \ i = 1, ..., n : \boldsymbol{\theta}_i = \boldsymbol{\theta}_j, \ \forall \ (i, j) \in \mathcal{E}\}.$$

We define the sum cost function as $f(\boldsymbol{\vartheta}) := (1/n)\sum_{i=1}^{n} f_i(\boldsymbol{\theta}_i)$, where $f : \mathcal{H} \to \mathbb{R} \cup \{\infty\}$ is a (possibly nonconvex) cost function of the ith agent. Equation (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 following equivalent optimization problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{\theta}).$$

The consensus formulation motivates a decentralized approach to finding high-quality solutions because each agent i can only access its local cost function f_i and process its local data through messages exchanged with its neighbors.

Without assuming convexity for (1), one cannot hope to find an optimal solution using a reasonable amount of effort, since solving a nonconvex problem is, in general, NP-hard [7]. Instead, we resort to finding stationary, consensual solutions whose gradients are small and variables are in consensus. Formally, letting $\epsilon \geq 0$, we say that $\boldsymbol{\vartheta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$ is an ϵ-stationary solution to (1) if

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

where $\bar{\boldsymbol{\theta}} := 1/n \ \Sigma_{i=1}^{n} \boldsymbol{\theta}_i$.

Next, we summarize two commonly used conditions when approaching (1).

Assumption 1

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

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

Furthermore, the average function $f : \mathcal{H} \to \mathbb{R} \cup \{\infty\}$ is lower-bounded across 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 second-order stationary and global optimal solutions. However, establishing these results requires additional assumptions, which further restricts the problem class. We refrain from discussing these works in detail due to space limitations.

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

Data model

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

Batch data

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

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

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

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, $f_i(\boldsymbol{\theta}_i) = \mathbb{E}_{\xi_i \sim \pi_i(\cdot)}[F_i(\boldsymbol{\theta}_i; \xi_i)]$, 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

$$\mathsf{DO}_i(\boldsymbol{\theta}_i^t) = \frac{1}{m_t}\sum_{\ell=1}^{m_t} \nabla F_i(\boldsymbol{\theta}_i^t; \xi_{i,\ell}^{t+1}), \tag{5}$$

where $\xi_{i,\ell}^{t+1} \sim \pi_i(\cdot)$, $\ell = 1,\ldots,m_t$, which is an unbiased estimate of the gradient; i.e., $\mathbb{E}_{\xi\sim\pi_i(\cdot)}[\mathsf{DO}_i(\boldsymbol{\theta})] = \nabla f_i(\boldsymbol{\theta})$.

Assumption 2

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

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

In other words, the random variable $\mathsf{DO}_i(\boldsymbol{\theta})$ has a uniformly bounded variance. A related setting involves a large but fixed data set ($M_i \gg 1$) at agent i, denoted by $\{\xi_{i,1},\ldots,\xi_{i,M_i}\}$, and f_i is given by (4). Accessing the full data set entails an undesirable $O(M_i)$ computation complexity. As a remedy, we can uniformly draw at each iteration a small batch of random samples ($m \ll M_i$) from the large data set. This results in a DO akin to (5).

Examples and challenges

We conclude this section by listing a few popular examples of nonconvex learning problems and how they fit into the models described previously. Moreover, we discuss the challenges with nonconvex distributed learning that motivate the algorithms to be reviewed in this article.

Example 1: Binary classifier training with neural network

For each $i \in \{1,\ldots,n\}$, suppose that a stream of training data $\xi_{i,1}^t, \xi_{i,2}^t, \ldots$ 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 $y_{i,j}^t \in \{0,1\}$. Letting $\boldsymbol{\theta} = (\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(L)})$ be the parameters of an L-layer neural network, we consider the model in (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 softmax 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 $\ell = 0,\ldots,L-1$, 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 for the neural network, taking into account the (potentially heterogeneous) data received at all agents. Here, the loss function $f_i(\boldsymbol{\theta})$ is nonconvex 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 by $\xi_{i,\ell} = \boldsymbol{x}_{i,\ell} \in \mathbb{R}^{m_1}$. The data received at the agents can be encoded using the columns of 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 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

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

where $\mathcal{A}, \mathcal{Y}_i$ 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 is given by: $f_i(\boldsymbol{\Phi}) = \min_{\boldsymbol{Y}_i\in\mathcal{Y}_i} \|\boldsymbol{X}_i - \boldsymbol{\Phi}\boldsymbol{Y}_i\|_{\mathrm{F}}^2$. The common variable is jointly decided by the data received at the agents, while the private variables are nuisance parameters determined locally. Here, the cost function $f_i(\boldsymbol{\theta})$ satisfies Assumption 1, and the DO follows the batch-data model.

Handling nonconvex distributed learning problems involves several unique challenges. First, directly applying algorithms developed for convex problems to the nonconvex setting may lead to unexpected algorithm behaviors. To see this, we provide a simple example below.

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 nonconvex. 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 the "Algorithms for Batch Data" section; see (18)] with the 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 iteration always diverges. On the contrary, we can verify that the 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, the 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 when their distributions vary between agents. 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 the 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 implementing distributed algorithms so they scale to large networks and model sizes. Moreover, an effective distributed algorithm should jointly design the communication and computation protocols. Addressing these challenges is the next focus.

Balancing communication and computation in distributed learning

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

capability of a distributed learning system. This section delineates how existing algorithms overcome such a challenge by using different data oracles. In a nutshell, the batch-data setting can be tackled through a primal-dual optimization framework or a family of gradient-tracking (GT) methods, while the streaming-data setting is commonly resolved through the DGD or GT approaches. A summary of the reviewed algorithms can be found in Figure 2. 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 (excluding self-loops). 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 $k \in \mathcal{V} \backslash \{i, j\}$. Note that $\boldsymbol{A}^\top \boldsymbol{A} := \boldsymbol{L}_G \in \mathbb{R}^{n \times n}$ is the graph Laplacian matrix. Finally, a mixing matrix, $\boldsymbol{W}$, satisfies the following conditions:

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

For instance, the mixing matrix can be chosen as the doubly stochastic matrix: For some $\alpha \in (0, 1)$,

$$W_{ij} = \begin{cases} \alpha / \max\{d_i, d_j\} & \text{if } (i, j) \in \mathcal{E}, \\ (1 - \alpha) + \sum_{j \in N_i} \max\{0, \alpha/d_i - \alpha/d_j\} & \text{if } i = j, \\ 0 & \text{if } (i, j) \notin \mathcal{E}; \end{cases} \tag{11}$$

see [14] for other designs. For any $\boldsymbol{\vartheta} \in \mathbb{R}^n$, we observe that $\boldsymbol{A}^\top \boldsymbol{A}\, \boldsymbol{\vartheta}, \boldsymbol{W}\boldsymbol{\vartheta}$ can be calculated via message exchange among neighboring agents.

The mixing and/or graph Laplacian matrix specifies the communication pattern in distributed learning. Since $\boldsymbol{W}^\infty = (1/n)\boldsymbol{1}\boldsymbol{1}^\top$,

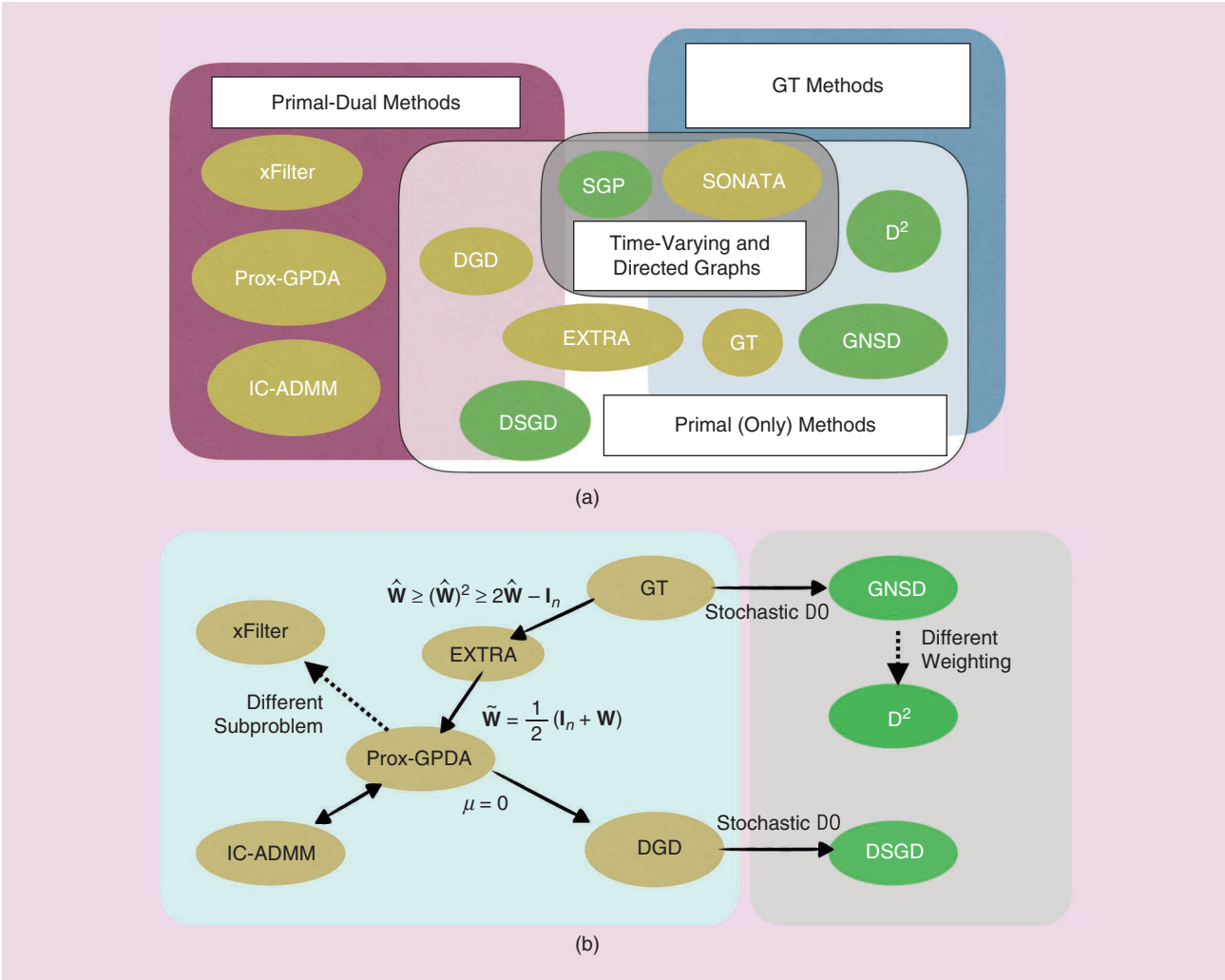


FIGURE 2. An overview of distributed algorithms for nonconvex learning. (a) The reviewed algorithms. (b) The connections between algorithms. Orange (respectively, green) patches refer to algorithms designed for batch (respectively, streaming) data. Lines with a single arrow indicate that one algorithm can reduce to another; those with double arrows mean that the algorithms are equivalent. Dotted lines signify that the algorithms are related (the conditions are given above the line). Prox-GDPA: proximal gradient primal-dual algorithm; IC-ADMM: inexact consensus, alternating-direction method of multipliers; DSGD: distributed stochastic GD; SGP: stochastic gradient push; EXTRA: exact first-order algorithm; SONATA: successive convex-approximation algorithm over time-varying digraphs; D^2: disjunctive decomposition; GNSD: GT-based nonconvex stochastic algorithm for decentralized training.

the mixing matrix enables one to distributively compute the average by repeatedly applying the mixing matrix. Because the gradient $(1/n)\Sigma_{i=1}^{n}\nabla f_i(\theta)$ is the average of the local gradients, the easiest way to derive a distributed algorithm is to compute the exact average of the gradient by applying $\boldsymbol{W}$ repeatedly. Such a distributed algorithm will behave exactly the same as the centralized gradient algorithm, and it may save computation (gradient or data-oracle evaluation) through a faster convergence rate; however, the communication (message exchange) cost can be overwhelming.

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 any time. This setting is typical for small-to-moderate data sets 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

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

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

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

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

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

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

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

$$\begin{aligned} \theta_i^{t+1} &\leftarrow \frac{1}{\beta_i + 2cd_i}\Big\{\beta_i\theta_i^t - \nabla f_i(\theta_i^t) - p_i^t + c\sum_{j|(i,j)\in\mathcal{E}}(\theta_i^t + \theta_j^t)\Big\}, \\ p_i^{t+1} &\leftarrow p_i^t + c\sum_{j|(i,j)\in\mathcal{E}}(\theta_i^{t+1} - \theta_j^{t+1}), \\ &\text{for any } i = 1, \ldots, n. \end{aligned} \tag{16}$$

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

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

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

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

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

We see that the Prox-GPDA has various equivalent forms in (13)–(17). In the following, we show that a number of existing algorithms share communication/computation steps that are similar to the Prox-GPDA.

Decentralized GD algorithm

Initially proposed by [13] for convex problems, the decentralized GD algorithm is one of the most popular distributed algorithms. It uses the penalized problem $\min_{\boldsymbol{\vartheta}\in\mathbb{R}^n}\Sigma_{i=1}^{n} f_i(\theta_i) + (1/2\alpha)\|\boldsymbol{\vartheta}\|^2_{\boldsymbol{I}_n - \boldsymbol{W}}$, where $\alpha > 0$ is a penalty parameter, and applies the GD method with step size α to yield

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

We note that this is a primal method because it apparently does not involve dual variables. Nonetheless, the decentralized GD's update formula can be derived from the 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, the decentralized GD requires a diminishing step size since $\alpha^t = 1/t$ and an additional assumption that $\nabla f_i(\theta_i)$ is bounded for any θ_i and $i = 1, \ldots, n$ [17, Th. 2].

EXTRA

The exact first-order algorithm (EXTRA) was proposed in [18] as an alternative to the decentralized GD, with a convergence guarantee achieved by 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 EXTRA is given by

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

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

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

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

Rate-optimal schemes

A fundamental question about the distributed problem in (1) is: What are the minimum computation and communication costs required to find an ϵ-stationary solution? An answer is given in [19]. For any distributed algorithm using gradient information, at least $\Omega((\epsilon\sqrt{\xi(\boldsymbol{L}_G)})^{-1})$ communication rounds and $\Omega(\epsilon^{-1})$ rounds of gradient evaluation are required to attain an ϵ-stationary solution, where $\xi(\boldsymbol{L}_G) := \lambda_{\min}(\boldsymbol{L}_G)/\lambda_{\max}(\boldsymbol{L}_G)$ is the ratio between the smallest nonzero and largest eigenvalues of the graph Laplacian matrix, $\boldsymbol{L}_G$. [In every gradient evaluation, each local node i evaluates $\nabla f_i(\cdot)$ once.] Interestingly, the Prox-GPDA, EXTRA, and IC-ADMM achieve the lower communication and computation bounds in star and fully connected networks. For the general network topology, [19] proposed a near-optimal scheme, xFilter, which updates $\boldsymbol{\vartheta}$ by considering

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

Compared to (13), the quadratic term $(c/2)\|\boldsymbol{A}\boldsymbol{\vartheta}\|^2$ 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 $Q = \tilde{O}(1/\sqrt{\xi(\boldsymbol{L}_G)})$ suffices to produce an algorithm that requires $\tilde{O}((\epsilon\sqrt{\xi(\boldsymbol{L}_G)})^{-1})$ communication rounds and $O(\epsilon^{-1})$ gradient-evaluation rounds, where the notation $\tilde{O}(\cdot)$ hides a log function of n (which is usually small). This matches the aforementioned lower bounds. Similarly, the communication effort required is nearly optimal (up to a multiplicative logarithmic factor). Further, compared with the other batch methods reviewed in this article, this is the only algorithm whose gradient-evaluation complexity is independent of the graph structure.

GT-based methods

Another class of algorithms that can deal with the nonconvex problem in (1) leverages the GT technique. The method is based on the simple idea that if every agent has access to the global gradient $1/n\,\Sigma_{i=1}^n \nabla f_i(1/n\,\Sigma_{j=1}^n \theta_j^t)$, the (centralized) GD can be performed at each agent. The GT technique provides an iterative approach to do so approximately. The algorithm performs two message exchanges during each iteration [with $\hat{\boldsymbol{W}}$ satisfying (10)]:

$$\begin{aligned}\boldsymbol{\vartheta}^{t+1} &\leftarrow \hat{\boldsymbol{W}}\boldsymbol{\vartheta}^t - \alpha\boldsymbol{g}^t,\\ \boldsymbol{g}^{t+1} &\leftarrow \hat{\boldsymbol{W}}\boldsymbol{g}^t + \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^{t+1}) - \nabla \boldsymbol{f}(\boldsymbol{\vartheta}^t),\ \forall\, t = 1, 2, \ldots.\end{aligned} \tag{22}$$

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], GT algorithm (22) converges at a rate of $O(1/\epsilon)$ 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 the 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

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

It can be shown that if $\hat{\boldsymbol{W}}$ satisfies $\hat{\boldsymbol{W}} \succeq (\hat{\boldsymbol{W}})^2 \succeq 2\hat{\boldsymbol{W}} - \boldsymbol{I}_n$, the algorithm takes the same form as the generalized EXTRA discussed after (20); see [21, Sec. 2.2.1]. However, the analysis for the generalized EXTRA works only on convex problems and does not carry over to the GT method in the nonconvex setting. We remark that all of the previous algorithms except the decentralized GD converge for Example 3. This is because for the latter example, the gradient can be unbounded.

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 for processing large-to-infinite data sets. Balancing the communication and computation costs is an important issue since even the centralized algorithm may have slow convergence. The first study of distributed stochastic algorithms dates to Tsitsiklis et al. [22] who studied the asymptotic convergence of the DSGD algorithm reviewed here. The DSGD algorithm is relevant to the distributed estimation problem that is important in adaptive signal processing;

therefore, many works study its transient behavior (of bias, mean-squared error, and so forth); e.g., [23] and [24] and the overview in [5]. Unfortunately, those works mainly focus on convex problems. We review the more recent results dedicated to the nonconvex learning setting with nonasymptotic convergence analysis.

DSGD algorithm

This class of algorithm replaces the deterministic oracle in the decentralized GD with the stochastic oracle (5). It takes the following form:

$$\boldsymbol{\vartheta}^{t+1} \leftarrow \boldsymbol{W}\boldsymbol{\vartheta}^{t} - \alpha^{t}\mathsf{DO}(\boldsymbol{\vartheta}^{t}),\ \forall\, t = 1,2,..., \tag{24}$$

where $\alpha^t > 0$ is the step size and $\mathsf{DO}(\boldsymbol{\vartheta}^t) := (\mathsf{DO}_1(\theta_1^t), \ldots, \mathsf{DO}_n(\theta_n^t))^\top$. Obviously, the DSGD can be implemented in a distributed manner via mixing matrix $\boldsymbol{W}$. The study of such an algorithm in the nonconvex setting dates to the work in [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 almost certainly converges to a first-order stationary solution. However, [22] mainly provides asymptotic convergence conditions without a clear indication of whether the DSGD can outperform its centralized counterpart.

Recently, the DSGD (as well as decentralized, parallel stochastic GD) was applied in [3] for the decentralized training of neural networks, and the convergence rate was analyzed in [25]. In the analysis by [25], the following condition for the data across agents is assumed:

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

Such an assumption can be difficult to verify, and it is required only when analyzing the DSGD convergence rate for nonconvex problems. For example, if the loss function is a quadratic function of the form, $f_i(\theta) = 1/2\left|a_i\theta + b_i\right|^2$, the corresponding gradient is a linear function of θ: $\nabla f_i(\theta) = a_i\theta + b_i$. The left-hand side 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 Assumptions 1 and 2, for any sufficiently large T, if we set $\alpha^t = O(\sqrt{n/(\sigma^2 T)})$ for all $t \ge 0$, the DSGD finds an approximate stationary solution to (1), satisfying $\mathbb{E}[\mathsf{Gap}(\vartheta^{\tilde{t}})] = O(\sigma/\sqrt{nT})$, where $\tilde{t}$ is uniformly drawn from $\{1, \ldots, T\}$ [25, Cor. 2]. Compared to the centralized SGD algorithm where a single sample is used each time, a speedup factor of $1/\sqrt{n}$ is observed, which is due to the variance-reduction effect that results from averaging the samples from the n nodes. Yet achieving this requires $\varsigma^2 = O(1)$ so that the data are homogeneous across the agents. See [11] and [12], which show that the DSGD algorithm converges to a second-order stationary solution under a condition similar to (25). In summary, the DSGD algorithm is simple to implement, but it has a major limitation when dealing with heterogeneous data, a shortcoming that is demonstrated in our numerical experiments.

D^2 algorithm

To remove the local data assumption in (25) from the DSGD, the D^2 algorithm was proposed in [26]. Using mixing matrix $\boldsymbol{W}$, the recursion of D^2 is given as

$$\boldsymbol{\vartheta}^{t+1} \leftarrow 2\boldsymbol{W}\boldsymbol{\vartheta}^{t} - \boldsymbol{W}\boldsymbol{\vartheta}^{t-1} - \alpha^{t}\boldsymbol{W}(\mathsf{DO}(\boldsymbol{\vartheta}^{t}) - \mathsf{DO}(\boldsymbol{\vartheta}^{t-1})),\quad \forall\, t = 1,2,\ldots. \tag{26}$$

In addition to the previous conditions for the weight matrix in (10), the D^2 requires a special condition, $\lambda_{\min}(\boldsymbol{W}) > -1/3$, which implies that the weight of combining the current node is greater than that of combining its neighbors. Together with Assumptions 1 and 2, for any sufficiently large T, we set $\alpha^t = O(\sqrt{n/(\sigma^2 T)})$ for all $t \ge 0$, and the D^2 finds an approximate stationary solution [26] to (1), satisfying $\mathbb{E}\left[\left\|n^{-1}\sum_{j=1}^{n}\nabla f_j(\vartheta^{\tilde{t}})\right\|^2\right] = O(\sigma/\sqrt{nT})$, where $\tilde{t}$ is uniformly drawn from $\{1, \ldots, T\}$.

Comparing (26) with (23) reveals a close similarity between the D^2 and GT: Both algorithms use the current and the previous DOs, 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 mixing matrix $\boldsymbol{W}$ for the D^2, and a $\boldsymbol{W}$ multiplies the difference of the gradient estimates. Such a seemingly minor difference turns out to be one major limiting factor for the D^2.

Example 4

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 that the D^2 diverges for any constant $\alpha^t \le 0.25$ or diminishing step size $\alpha^t = 1/t$.

Distributed stochastic GT algorithm

How can we design algorithms that can deal with heterogeneous data and require conditions weaker than that of D^2? The GNSD algorithm was proposed in [28]; essentially, it is a stochastic version of the GT method in (23):

$$\boldsymbol{\vartheta}^{t+1} \leftarrow 2\boldsymbol{W}\boldsymbol{\vartheta}^{t} - \boldsymbol{W}^2\boldsymbol{\vartheta}^{t-1} - \alpha^{t}[\mathsf{DO}(\boldsymbol{\vartheta}^{t}) - \mathsf{DO}(\boldsymbol{\vartheta}^{t-1})],\ \forall\, t = 1,2,\ldots. \tag{27}$$

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

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

Other distributed algorithms

Despite the differences in the DOs used and assumptions needed for convergence, the reviewed algorithms may be regarded

as variants of unconstrained GD methods for a single parameter (vector) on a fixed communication graph. However, special communication and computation architectures may arise in practice. We conclude the section by highlighting a few works in relevant directions.

Coordinate descent methods

When the optimization model in (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 constant. The matrix factorization problem discussed in Example 2 is one such instance. Specifically, [15] and [29] respectively propose to combine the Prox-GPDA and GT with coordinate descent to tackle the distributed dictionary-learning problem (batch data), with some convergence guarantees.

Directed and time-varying graphs

We have assumed that the graph connecting the agents is undirected and static. However, directed and time-varying graph topology may arise in practice; e.g., with unreliable networks. Several works have been proposed for various settings [20], [30], [31]. For batch data, [20] suggested the SONATA algorithm, which combines GT with the push-sum technique; for streaming data, [31] offered the SGP algorithm, which incorporates the SGD and push-sum approach. SONATA and SGP are shown to converge sublinearly to a stationary solution on time-varying and directed graphs.

Practical issues and numerical results

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

Our experiments are conducted on two computer clusters, one provided by the Minnesota Supercomputing Institute (MSI), the other by Amazon Web Services (AWS). The MSI cluster has better independent computation power at each node but a worse communication bandwidth than the AWS cluster; see Figure 3(b). The MSI nodes have Intel Haswell E5-2680v3 CPUs at 3.2 GHz and 14-gigabytes/s internode communication, while the AWS nodes have Intel Xeon E5-2686v4 CPUs at 3 GHz, NVIDIA K80 GPUs, and 25-gigabytes/s internode communication.

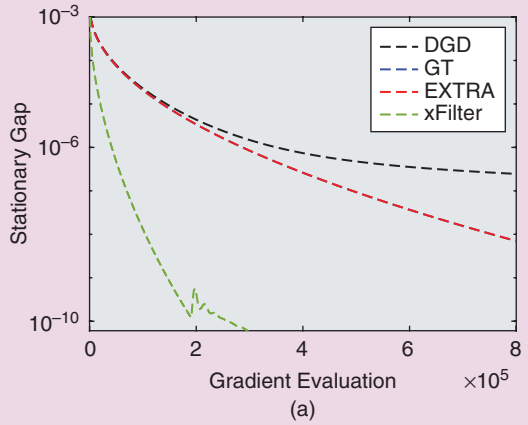
Two sets of experiments are conducted. The first compares different algorithms on a single machine. Since the distributed implementation is only simulated, the purpose of this set is to understand the algorithms' theoretical behavior. The second set showcases the algorithm performance on truly distributed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

Experiment set 1

We consider tackling a regularized logistic regression problem with a nonconvex regularizer in a distributed manner. We use notations similar to those in Example 1; i.e., the feature is x_i^ℓ, and the label is y_i^ℓ. Letting $\lambda,\ \rho > 0$ be the regularizer's parameters, each local cost function f_i is given by

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

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 of degree five. The stationarity gap versus the number of gradient evaluation for the surveyed batch algorithms is shown in Figure 3(a). In terms of the number of full gradient evaluations, the xFilter is the fastest.



(a)

Cost per Iteration	Computation (m)				Communication (n)		
Settings	128	8	64	256	2	8	32
MSI, DSGD	1				0.3	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	1.6	4.21
MSI, DSGD		1	1.09	1.36			2.61
MSI, GNSD		1	1.12	1.45			8.47

(b)

FIGURE 3. (a) The stationarity gap against the iteration number of different algorithms with a synthetic data set and n = 32 agents. Note that the curves for the GT and EXTRA overlap. (b) The 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. DGD: decentralized gradient descent.

Experiment set 2

We focus on the DSGD and GNSD algorithms for streaming data and apply them to train a neural network, as in Example 1, and classify handwritten digits from the Modified National Institute of Standards and Technology (MNIST) data set. The neural network contains two hidden layers with 512 and 128 neurons each and 4.68×10^5 parameters. 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 the "Problems and Data Models" section, and all agents use the same minibatch sizes $m_t = m$. The communication graph is a random regular graph with degree five.

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

Network scalability

We analyze how the network size n affects the overall convergence speed. Intuitively, if the communication cost is relatively lower than that of computation, it is beneficial to use a larger network and involve more agents to share the computational burden. In Figure 4(a) and (b), we see that the runtime performance of the DSGD/GNSD algorithms on AWS significantly improves as 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 Figure 4(a) the benefit of distributed learning ($n > 1$) compared with the centralized scheme ($n = 1$), where the DSGD with multiple agents can reach a smaller optimality gap. On both platforms, we observe that the GNSD achieves an even smaller optimality gap compared with the DSGD but requires more time to complete the given number of epochs. This is reasonable because, as discussed in the "Algorithms for Streaming Data" section, the DSGD requires one round of communication per DO evaluation, while the GNSD requires two.

Graph topology

Another key parameter that has 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 a large $\xi(\boldsymbol{L}_G)$] have a faster convergence rate, factors such as the maximum degree of the agents also matter. In Figure 4(b), we compare the runtime with $n = 32$ agents on different types of topology, including a complete graph, random regular graph with degree five, hypercube graph, and circle graph. We observe that well-connected sparse graphs (e.g., random regular and hypercube) are preferred, since there are fewer communication overheads compared with dense graphs (e.g., the complete graph) and poorly-connected graphs (e.g., the circle graph).

Minibatch size

The choice of minibatch size m is another important parameter. While it speeds up the convergence with a large minibatch size, it can be computationally expensive and requires extensive memory. We examine the tradeoff with the minibatch size in Figure 5(a), where the experiments are run on the MSI cluster. Increasing the batch size improves the GNSD algorithm more

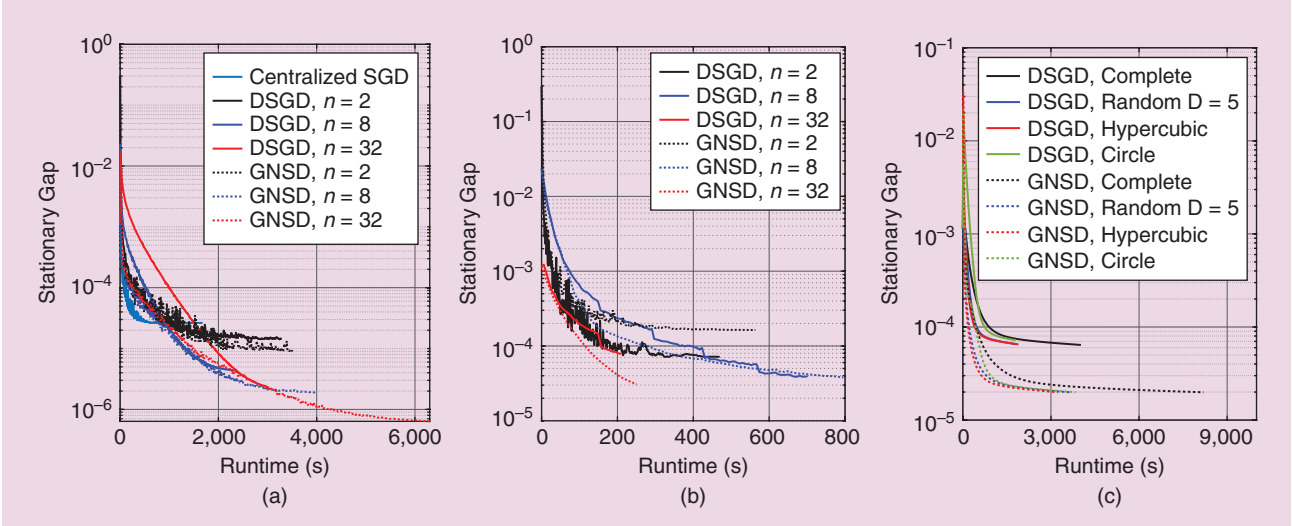


FIGURE 4. The runtime comparison of streaming algorithms on (a) MSI with $n = 1, 2, 8, 32$ agents and batch size $m = 128$ for all algorithms, terminated in 450 epochs; (b) AWS with $n = 2, 8, 32$ agents and batch size $m = 128$, terminated in 128 epochs; and (c) MSI with different types of graph topologies with $n = 32$ agents and batch size $m = 128$, terminated in 256 epochs.

significantly than the DSGD does. In the lower part of Figure 3(b), we provide the normalized per-iteration computation and communication times for different minibatch sizes. Notice that for the DSGD, it takes 1.09 and 1.36 units of computation time with minibatch sizes of $m = 64$ and $m = 256$, compared to the baseline setting with $m = 8$. A larger minibatch size seems to be more efficient.

Heterogeneous data

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

Model size

Intuitively, small models may benefit from distributed algorithms because there is a modest amount of information to exchange, especially in systems where the communication is slower than the computation. As shown in Figure 5(b), we compare three neural networks—a small network (a two-layer fully connected neural network with 8×10^3 parameters), a medium network (LeNet-5 with two convolutional layers, three fully connected layers, and 6×10^4 parameters), and a large network (the Keras example for the MNIST, with four convolutional layers, three fully connected layers, and 4.07×10^5 parameters)—that run on the MSI cluster with the DSGD. As the model size increases, the communication-cost growth outweighs the computation cost.

Related issues

Another active research area relates to improving the communication efficiency in distributed algorithms. Taking the DSGD as an example, a possible idea is to perform SGD updates multiple times (say, I) at an agent before exchanging the parameters with the neighbors. Using this scheme, [32] shows that with $I = \Theta(1/\epsilon)$, the distributed algorithm run on a star-graph topology requires only $O(1/\epsilon)$ [respectively, $O(1/\epsilon^{3/2})$] message exchanges for a homogeneous (respectively, heterogeneous) data set to find an ϵ-stationary solution to (1). Alternatively, [33] proposes to skip unnecessary communication steps when the deviation of the local variables is small. Lastly, to reduce the time expense to synchronize across agents and make distributed learning less vulnerable to straggling agents, there are works that enable asynchronous communication; see [31] and [34] for examples.

Conclusions and open problems

This article reviewed some selected developments of nonconvex distributed learning algorithms. It showed the interplay among problems, data, and computation and communication, leading to different algorithms. These algorithms are compared by using numerical experiments on computer clusters to show their practical potential. To conclude, we list a few directions for future research.

Dynamic data

Beyond batch and streaming data, an open problem relates to developing distributed algorithms for dynamic data. We consider a DO that takes the same form as the first equation in (5), but the data samples $\{\xi_{i,\ell}^{t+1}\}_{\ell=1}^{M_i}$ are drawn, instead, from 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. 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

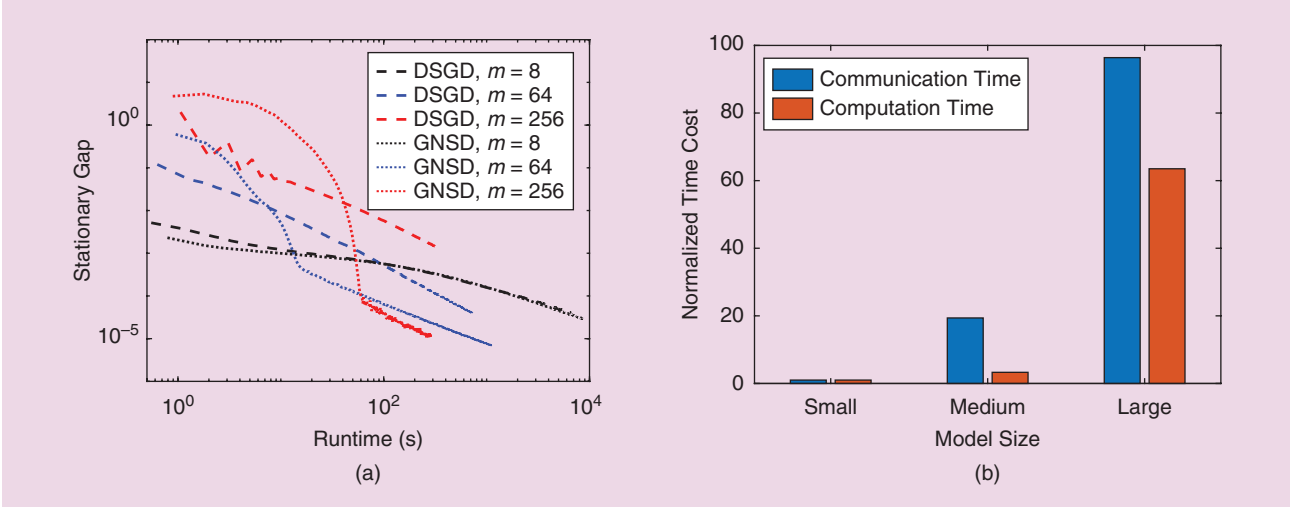


FIGURE 5. (a) The runtime comparison of minibatch size $m = 8, 64, 256$ on MSI, terminated after 256 epochs. The data are heterogeneous where each node is assigned exclusive classes. (b) The small-model normalized computation and communication cost for different model sizes on MSI with $n = 32$ agents.

generated through interactions with the environment (and therefore dependent on the current policy $\boldsymbol{\vartheta}^t$); the state/action pairs will be used to compute the policy gradient for updating $\boldsymbol{\vartheta}^{t+1}$.

Distributed algorithms based on the dynamic DO are challenging to analyze since the computation, communication, and data acquisition have to be jointly considered. To the best of our knowledge, such a setting has been studied only recently for a centralized algorithm in [35]. In a distributed setting, progress has been made in multiagent reinforcement learning; for instance, [36] applied a linear function approximation to simplify the nonconvex learning problem as a convex one. Nevertheless, a truly distributed, nonconvex algorithm with a dynamic DO has been neither proposed nor analyzed. Another challenging dynamic scenario concerns 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 [37].

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 constitute private records that cannot be shared. This imposes the challenging question of how to enable the agents that own different sets of features to collaborate on the learning task; see [16] and [38].

Federated and robust learning

To improve user privacy, federated learning (FL) 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 critical; for example, [39].

Distributed learning with statistical guarantees

The algorithms surveyed in this article aim to compute high-quality solutions so that optimization-based conditions, such as (2), are satisfied. It is also interesting to investigate whether these algorithms can achieve strong statistical guarantees for specific ML problems, such as nonconvex M-estimation [40], so that ground-truth parameters can also be recovered.

Acknowledgments

We would like to thank the anonymous reviewers as well as Dr. Gesualdo Scutari and Dr. Angelia Nedić for helpful comments that significantly improved the quality of the article. Tsung-Hui Chang was supported, in part, by the National Key R&D Program of China (grant 2018YFB1800800), National Natural Science Foundation of China (grant 61731018), and Shenzhen Fundamental Research Fund (grants JCYJ20190813171003723 and KQTD2015033114415450). Hoi-To Wai was supported by the Chinese University of Hong Kong (direct grant 4055113). Mingyi Hong, Songtao Lu, and Xinwei Zhang were supported, in part, by the National Science Foundation (grants CMMI-172775 and CIF-1910385) and Army Research Office (grant 73202-CS). This work was done when Songtao Lu was a postdoctoral fellow at the University of Minnesota.

Authors

Tsung-Hui Chang (tsunghui.chang@ieee.org) is an associate professor in the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, as well as the Shenzhen Research Institute of Big Data, China. He has held research positions at the National Tsing Hua University, Hsinchu, Taiwan, and the University of California, Davis. His research interests include data communications and distributed optimization and its applications. He received the IEEE Communications Society Asia-Pacific Outstanding Young Researcher Award in 2015 and the IEEE Signal Processing Society Best Paper Award in 2018. He served or serves as associate editor of *IEEE Transactions on Signal Processing*, *IEEE Transactions on Signal and Information Processing Over Networks*, and *IEEE Open Journal of Signal Processing*. He is a member of the IEEE Signal Processing for Communications and Networking Technical Committee. He is a Senior Member of the IEEE.

Mingyi Hong (mhong@umn.edu) received his Ph.D. degree from the University of Virginia, Charlottesville, in 2011. He is an assistant professor in the Department of Electrical and Computer Engineering at the University of Minnesota, Minneapolis. He serves on the IEEE Signal Processing for Communications and Networking and Machine Learning for Signal Processing Technical Committees. His research interests include optimization theory and applications in signal processing and machine learning. He is a Member of the IEEE.

Hoi-To Wai (htwai@cuhk.edu.hk) received his Ph.D. degree in electrical engineering from Arizona State University (ASU), Tempe, and his B.Eng. and M.Phil. degrees in electronic engineering from the Chinese University of Hong Kong (CUHK). He is an assistant professor in the Department of Systems Engineering and Engineering Management at CUHK and previously held research positions at ASU; the University of California, Davis; Telecom ParisTech; Ecole Polytechnique; Laboratory for Information & Decision Systems; and the Massachusetts Institute of Technology. His research interests include signal processing, machine learning, and distributed optimization. His dissertation received the Dean's Dissertation Award from ASU, and he received a Best Student Paper Award at the International Conference on Acoustics, Speech, and Signal Processing. He is a Member of the IEEE.

Xinwei Zhang (zhan6234@umn.edu) received his B.S. degree in automation from the University of Science and Technology of China, Anhui, in 2018 and is pursuing his Ph.D. degree in the Electrical and Computer Engineering Department at the University of Minnesota, Minneapolis. His research interests include distributed optimization and power-system control. He is a Student Member of the IEEE.

Songtao Lu (songtao@ibm.com) received his Ph.D. degree in electrical engineering from Iowa State University, Ames, in 2018. He is a research scientist in the IBM Research Artificial Intelligence residency program at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and previously was a postdoctoral associate with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. He received the Graduate and Professional Student Senate Research Award from Iowa State University in 2015, Research Excellence Award from the Graduate College of Iowa State University in 2017, and Student Travel Awards from the 20th International Conference on Artificial Intelligence and Statistics in 2017 and 36th International Conference on Machine Learning. His research interests include artificial intelligence, optimization, signal processing, and machine learning. He is a Member of the IEEE.

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 *Proc. 31st Int. Conf. Neural Information Processing System*, 2017, pp. 1508–1518. doi: 10.5555/3294771.3294915.

[2] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, GossipGraD: Scalable deep learning using gossip communication based asynchronous gradient descent. 2018. [Online]. Available: arXiv:1803.05880

[3] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *Proc. 31st Int. Conf. Neural Information Processing System*, 2017, pp. 5906–5916. doi: 10.5555/3295222.3295340.

[4] A. Nedić and A. Ozdaglar, "Cooperative distributed multi-agent optimization," in *Convex Optimization in Signal Processing and Communications,* D. P. Palomar and Y. C. Eldar, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2010, pp. 340–386.

[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. doi: 10.1109/MSP.2012.2231991.

[6] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 32–43, 2014. doi: 10.1109/MSP.2014.2329397.

[7] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Math. Program.*, vol. 39, no. 2, pp. 117–129, June 1987. doi: 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 *Proc. 35th Int. Conf. Machine Learning*, 2018, pp. 2009–2018.

[9] A. Daneshmand, G. Scutari, and V. Kungurtsev, Second-order guarantees of distributed gradient algorithms. 2018. [Online]. Available: arXiv:1809.08694

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

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

[12] S. Vlaski and A. H. Sayed, Distributed learning in non-convex environments–part II: Polynomial escape from saddle-points. 2019. [Online]. Available: arXiv:1907.01849

[13] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, 2009. doi: 10.1109/TAC.2008.2009515.

[14] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing Markov chain on a graph," *SIAM Rev.*, vol. 46, no. 4, pp. 667–689, 2004. doi: 10.1137/S0036144503423264.

[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 *Proc. 34th Int. Conf. Machine Learning*, 2017, pp.1529–1538. doi: 10.13140/RG.2.2.25204.14729.

[16] T.-H. Chang, M. Hong, and X. Wang, "Multi-agent distributed optimization via inexact consensus ADMM," *IEEE Trans. Signal Process.*, vol. 63, no. 2, pp. 482–497, Jan 2015. doi: 10.1109/TSP.2014.2367458.

[17] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Trans. Signal Process.*, vol. 66, no. 11, pp. 2834–2848, June 2018. doi: 10.1109/TSP.2018.2818081.

[18] W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. Optim.*, vol. 25, no. 2, pp. 944–966, 2014. doi: 10.1137/14096668X.

[19] H. Sun and M. Hong, "Distributed non-convex first-order optimization and information processing: Lower complexity bounds and rate optimal algorithms," *IEEE Trans. Signal Process.*, vol. 67, no. 22, pp. 5912–5928, July 2019. doi: 10.1109/TSP.2019.2943230.

[20] G. Scutari and Y. Sun, "Distributed nonconvex constrained optimization over time-varying digraphs," *Math. Program.*, vol. 176, nos. 1–2, pp. 497–544, 2019. doi: 10.1007/s10107-018-01357-w.

[21] A. Nedić, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM J. Optim.*, vol. 27, no. 4, pp. 2597–2633, 2017. doi: 10.1137/16M1084316.

[22] J. Tsitsiklis, D. P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Autom. Control*, vol. 31, no. 9, pp. 803–812, 1986. doi: 10.1109/TAC.1986.1104412.

[23] F. S. Cattivelli and A. H. Sayed, "Diffusion LMS strategies for distributed estimation," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1035–1048, 2009. doi: 10.1109/TSP.2009.2033729.

[24] S. Kar, J. M. Moura, and K. Ramanan, "Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication," *IEEE Trans. Inf. Theory*, vol. 58, no. 6, pp. 3575–3605, 2012. doi: 10.1109/TIT.2012.2191450.

[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 *Proc. 31st Int. Conf. Neural Information Processing Systems*, 2017, pp. 5330–5340. doi: 10.5555/3295222.3295285.

[26] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D^2: Decentralized training over decentralized data," in *Proc. 35th Int. Conf. Machine Learning*, July 10–15, 2018, pp. 4848–4856.

[27] J. Zhang and K. You, Decentralized stochastic gradient tracking for empirical risk minimization. 2019. [Online]. Available: arXiv:1909.02712

[28] S. Lu, X. Zhang, H. Sun, and M. Hong, "GNSD: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization," in *Proc. IEEE Data Science Workshop (DSW)*, June 2019, pp. 315–321. doi: 10.1109/DSW.2019.8755807.

[29] A. Daneshmand, Y. Sun, G. Scutari, F. Facchinei, and B. M. Sadler, "Decentralized dictionary learning over time-varying digraphs," *J. Mach. Learn. Res.*, vol. 20, pp. 1–62, Sept. 2019.

[30] R. Xin, A. K. Sahu, U. A. Khan, and S. Kar, "Distributed stochastic optimization with gradient tracking over strongly-connected networks," in *Proc. IEEE 58th Conf. Decision and Control (CDC)*, 2019.

[31] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. 36th Int. Conf. Machine Learning*, 2019, pp. 344–353.

[32] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization," in *Proc. 36th Int. Conf. Machine Learning*, 2019, pp. 7184–7193.

[33] R. Aragues, G. Shi, D. V. Dimarogonas, C. Sagues, and K. H. Johansson, "Distributed algebraic connectivity estimation for adaptive event-triggered consensus," in *Proc. American Control Conf. (ACC)*, June 2012, pp. 32–37. doi: 10.1109/ACC.2012.6315110.

[34] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. 35th Int. Conf. Machine Learning*, July 10–15, 2018, pp. 3043–3052.

[35] B. Karimi, B. Miasojedow, E. Moulines, and H.-T. Wai, "Non-asymptotic analysis of biased stochastic approximation scheme," in *Proc. Int. Conf. Learning Theory*, 2019, pp. 1944–1974.

[36] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Başar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. Int. Conf. Machine Learning*, 2018, pp. 9340–9371.

[37] S. Shahrampour and A. Jadbabaie, "Distributed online optimization in dynamic environments using mirror descent," *IEEE Trans. Autom. Control*, vol. 63, no. 3, pp. 714–725, 2017. doi: 10.1109/TAC.2017.2743462.

[38] Y. Hu, D. Niu, J. Yang, and S. Zhou, "FDML: A collaborative machine learning framework for distributed features," in *Proc. ACM Int. Conf. Knowledge Discovery & Data Mining*, Aug. 2018, pp. 2232–2240.

[39] Z. Yang and W. U. Bajwa, "Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 5, no. 4, pp. 611–627, 2019. doi: 10.1109/TSIPN.2019.2928176.

[40] P.-L. Loh and M. J. Wainwright, "Regularized m-estimators with nonconvexity: Statistical and algorithmic theory for local optima," *J. Mach. Learn. Res.*, vol. 16, no. 19, pp. 559–616, 2015.

[41] M. T. Jones, "Security and the IoT ecosystem: Implementing security during the design phase," IBM Developer, Mar. 26, 2018. [Online]. Available: https://developer.ibm.com/articles/se-iot-security/

SP