Gradient tracking and variance reduction for decentralized optimization and machine learning

Ran Xin, Soummya Kar, and Usman A. Khan

Abstract

Decentralized methods to solve finite-sum minimization problems are important in many signal processing and machine learning tasks where the data is distributed over a network of nodes and raw data sharing is not permitted due to privacy and/or resource constraints. In this article, we review decentralized stochastic first-order methods and provide a unified algorithmic framework that combines variance-reduction with gradient tracking to achieve both robust performance and fast convergence. We provide explicit theoretical guarantees of the corresponding methods when the objective functions are smooth and strongly-convex, and show their applicability to non-convex problems via numerical experiments. Throughout the article, we provide intuitive illustrations of the main technical ideas by casting appropriate tradeoffs and comparisons among the methods of interest and by highlighting applications to decentralized training of machine learning models.

I. Introduction

In multi-agent networks and large-scale machine learning, when data is available at different devices with limited communication, it is often desirable to seek scalable learning methods that do not require bringing, storing, and processing data at one single location. In this article, we describe decentralized, stochastic first-order methods, which are particularly favorable to such ad-hoc and resource-constrained settings. Specifically, we describe a unified algorithmic framework for combining different ***variance reduction methods*** with ***gradient tracking*** in order to significantly improve upon the performance of the standard decentralized stochastic gradient descent (DSGD). However, this improvement comes at a price of losing the simplicity of DSGD and we study the added communication, computation, and storage requirements with the help of precise technical statements. For the ease of accessibility, we restrict the theoretical arguments to smooth and strongly-convex objectives, while the applicability to non-convex problems is shown with the help of numerical experiments. We emphasize that smooth and strongly-convex objectives are relevant in many machine learning applications, e.g., problems where a strongly-convex regularization is added to otherwise convex costs, or problems where the objective functions are non-convex but strongly-convex in the neighborhood of the local minimizers [1]. To provide context, we start by briefly reviewing the problems of interest and their associated centralized solutions.

A. Empirical Risk Minimization

In parametric learning and inference problems, the goal of a typical machine learning system is to find a model g, parameterized by a real vector $\boldsymbol{\theta} \in \mathbb{R}^p$, that maps an input data point $\mathbf{x} \in \mathbb{R}^{d_\mathbf{x}}$ to its corresponding output $\mathbf{y} \in \mathbb{R}^{d_\mathbf{y}}$. The setup requires defining a loss function $l(g(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}))$, which represents the loss incurred by the model g with parameter $\boldsymbol{\theta}$ on the data $(\mathbf{x}, \mathbf{y})$. In the formulation of statistical machine learning, we assume that each data sample $(\mathbf{x}, \mathbf{y})$ belongs to a joint probability distribution $\mathcal{P}(\mathbf{x}, \mathbf{y})$. Ideally, we would like to find the optimal model parameter $\widetilde{\boldsymbol{\theta}}^*$ by minimizing the following *risk (expected loss) function* $\widetilde{F}(\boldsymbol{\theta})$:

$$\text{P0:} \qquad \widetilde{\boldsymbol{\theta}}^* = \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\operatorname{argmin}} \widetilde{F}(\boldsymbol{\theta}), \qquad \widetilde{F}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{P}(\mathbf{x},\mathbf{y})} l(g(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})).$$

However, the true distribution $\mathcal{P}(\mathbf{x}, \mathbf{y})$ is often hidden or intractable in practice. In supervised machine learning, one usually has access to a large set of training samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, which can be considered as independent and identically distributed (i.i.d.) realizations from the distribution $\mathcal{P}(\mathbf{x}, \mathbf{y})$. The average of the losses incurred by the model $\boldsymbol{\theta}$ on a finite set of training data samples $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, known as the *empirical risk*, thus serves as an appropriate surrogate for the risk function $\widetilde{F}(\boldsymbol{\theta})$. Formally, the *empirical risk minimization* problem is stated as

$$\text{P1:} \qquad \boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathbb{R}^p}{\operatorname{argmin}} F(\boldsymbol{\theta}), \qquad F(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_{i=1}^{N} l(g(\boldsymbol{\theta}; \mathbf{x}_i, \mathbf{y}_i)) \triangleq \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta}^*$ is the minimizer of the empirical risk F. This finite-sum formulation captures a wide range of supervised learning problems. Examples include: hand-written character recognition with regularized logistic regression where the objective functions are smooth and strongly-convex [2]; text classification with support vector machines where the objectives are convex but not necessarily smooth [1]; and perception tasks with deep neural networks where the cost functions are non-convex in general [1], [2].

Our focus in this article is on smooth and strongly-convex objective functions defined as follows. An *L-smooth* and *μ-strongly-convex* function $f : \mathbb{R}^p \to \mathbb{R}$ is such that $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^p$ and for some positive constants $L, \mu > 0$, we have

$$\frac{\mu}{2} \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2^2 \leq f(\boldsymbol{\theta}_2) - f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_1)^\top (\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1) \leq \frac{L}{2} \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2^2.$$

We define $\mathcal{S}_{\mu,L}$ as the class of functions that are L-smooth and μ-strongly-convex [3]. We note that if $F \in \mathcal{S}_{\mu,L}$, then it has a unique global minimum denoted as $\boldsymbol{\theta}^*$. For any $F \in \mathcal{S}_{\mu,L}$, we have that $L \geq \mu$, and we define $\kappa \triangleq \frac{L}{\mu}$ as the condition number of F [3]; clearly, $\kappa \geq 1$. For the ease of accessibility, we restrict the theoretical arguments to the function class $\mathcal{S}_{\mu,L}$, while the applicability to non-convex problems is shown with the help of numerical experiments.

B. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet powerful method that has been extensively used to solve the empirical risk minimization problem P1. SGD, in its simplest form, starts with an arbitrary $\boldsymbol{\theta}_0 \in \mathbb{R}^p$ and performs the following iterations to learn $\boldsymbol{\theta}^*$ as $k \to \infty$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \cdot \nabla f_{s_k}(\boldsymbol{\theta}_k), \qquad k \geq 0, \tag{2}$$

where s_k is chosen uniformly at random from $\{1, \cdots, N\}$ and $\{\alpha_k\}_{k\geq 0}$ is a sequence of positive step-sizes. Comparing to batch gradient method where the descent direction $\nabla F(\boldsymbol{\theta}_k)$ at each iteration k is computed from the entire batch of data, SGD iteratively descends in the direction of the gradient of a randomly sampled component function. SGD is thus computationally-efficient as it evaluates one component gradient (extendable to more than one randomly selected functions) at each iteration and is a popular alternative in problems with a large number of high-dimensional training data samples and model parameters.

We note that the stochastic gradient $\nabla f_{s_k}(\boldsymbol{\theta}_k)$ is an unbiased estimate of batch gradient $\nabla F(\boldsymbol{\theta}_k)$, i.e., $\mathbb{E}_{s_k}[\nabla f_{s_k}(\boldsymbol{\theta}_k)|\boldsymbol{\theta}_k] = \nabla F(\boldsymbol{\theta}_k)$. Under the assumptions that each $f_i \in \mathcal{S}_{\mu,L}$ and each stochastic gradient $\nabla f_{s_k}(\boldsymbol{\theta}_k)$ has bounded variance[1], i.e., $\mathbb{E}_{s_k}\left[\|\nabla f_{s_k}(\boldsymbol{\theta}_k) - \nabla F(\boldsymbol{\theta}_k)\|_2^2 \,|\boldsymbol{\theta}_k\right] \leq \sigma^2, \forall k$, we note that with a constant step-size $\alpha \in \left(0, \frac{1}{L}\right]$, $\mathbb{E}\left[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|_2^2\right]$ decays linearly (on the log-scale), at the rate of $(1-\mu\alpha)^k$, to a neighborhood of $\boldsymbol{\theta}^*$. Formally, we have [1],

$$\mathbb{E}\left[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|_2^2\right] \leq (1-\mu\alpha)^k + \frac{\alpha\sigma^2}{\mu}, \qquad \forall k \geq 0. \tag{3}$$

This steady-state error $\frac{\alpha\sigma^2}{\mu}$ or the *inexact convergence* is due to the fact that $\nabla f_{s_k}(\boldsymbol{\theta}^*) \neq 0$, in general, and the step-size is constant. A diminishing step-size overcomes this issue and leads to an *exact convergence* to the minimizer $\boldsymbol{\theta}^*$ albeit at slower rate. For example, with $\alpha_k = \frac{1}{\mu(k+1)}$, we have

$$\mathbb{E}\left[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|_2^2\right] \leq \frac{\max\left\{\frac{2\sigma^2}{\mu^2}, \|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|_2^2\right\}}{k+1}, \tag{4}$$

for all $k \geq 0$, [1]. In other words, to reach an ϵ-accurate solution of $\boldsymbol{\theta}^*$, i.e., $\mathbb{E}\left[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2\right] \leq \epsilon$, SGD (with decaying step-sizes) requires $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ component gradient evaluations.

C. Variance-Reduced Stochastic Gradient Descent

In practice, a successful implementation of SGD relies heavily on the tuning of the step-sizes, and typically a decaying step-size sequence $\{\alpha_k\}_{k\geq 0}$ has to be carefully chosen due to the

[1]In this article, we restrict to the bounded variance assumption for simplicity. This assumption however can be relaxed, see [1], [4], [5], for example.

potentially large variance in SGD, i.e., the sampled gradient $\nabla f_{s_k}(\boldsymbol{\theta}_k)$ at $\boldsymbol{\theta}_k$ can be very far from the batch gradient $\nabla F(\boldsymbol{\theta}_k)$. In recent years, certain Variance-Reduction (VR) techniques have been developed towards addressing this issue [6]–[9]. The key idea here is to design an iterative estimator of the batch gradient whose variance progressively decays to zero as $\boldsymbol{\theta}_k$ approaches $\boldsymbol{\theta}^*$. Benefiting from this reduction in variance, VR methods have a low per-iteration computation cost, a key feature of SGD, and, at the same time, converge linearly to the minimizer $\boldsymbol{\theta}^*$ as the batch gradient descent (with a *constant* step-size for the objective function class $\mathcal{S}_{\mu,L}$). Different constructions of the aforementioned gradient estimator lead to different VR methods [6]–[9]. We focus on two popular VR methods in this article described as follows.

SAGA [7]: The SAGA method starts with an arbitrary $\boldsymbol{\theta}_0 \in \mathbb{R}^p$ and maintains a table that stores all component gradients $\{\nabla f_i(\widehat{\boldsymbol{\theta}}_i)\}_{i=1}^N$, where $\widehat{\boldsymbol{\theta}}_i$ denotes the most recent iterate at which ∇f_i was evaluated, initialized with $\{\nabla f_i(\boldsymbol{\theta}_0)\}_{i=1}^N$. At every iteration $k \geq 0$, SAGA chooses an index s_k uniformly at random from $\{1, \ldots, N\}$ and performs the following two updates:

$$\mathbf{g}_k = \nabla f_{s_k}(\boldsymbol{\theta}_k) - \nabla f_{s_k}(\widehat{\boldsymbol{\theta}}_{s_k}) + \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(\widehat{\boldsymbol{\theta}}_i), \qquad \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \cdot \mathbf{g}_k. \tag{5}$$

Subsequently, the entry $\nabla f_{s_k}(\widehat{\boldsymbol{\theta}}_{s_k})$ in the gradient table is replaced by $\nabla f_{s_k}\big(\boldsymbol{\theta}_k\big)$, while the other entries remain unchanged. Under the assumption that each $f_i \in \mathcal{S}_{\mu,L}$, it can be shown that with $\alpha = \frac{1}{3L}$, we have [7],

$$\mathbb{E}\left[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|_2^2\right] \leq C\left(1 - \min\left\{\frac{1}{4N}, \frac{1}{3\kappa}\right\}\right)^k, \qquad \forall k \geq 0, \tag{6}$$

for some $C > 0$. In other words, SAGA achieves ϵ-accuracy of $\boldsymbol{\theta}^*$ with $\mathcal{O}\left(\max\{N, \kappa\} \log \frac{1}{\epsilon}\right)$ component gradient evaluations, where recall that $\kappa = \frac{L}{\mu}$ is the condition number of the global objective function F. Indeed, SAGA has a non-trivial storage cost of $\mathcal{O}\left(Np\right)$ due to the gradient table, which can be reduced to $\mathcal{O}(N)$ for certain problems of interest, for example, logistic regression and least squares, by exploiting the structure of the objective functions [6], [7].

SVRG [8]: Instead of storing the gradient table, SVRG achieves variance reduction by computing the batch gradient periodically and can be interpreted as a *double-loop* method described as follows. The outer loop of SVRG, indexed by k, updates the estimate $\boldsymbol{\theta}_k$ of $\boldsymbol{\theta}^*$. At each outer iteration k, SVRG computes the batch gradient $\nabla F(\boldsymbol{\theta}_k)$ and executes a finite number T of SGD-type inner loop iterations, indexed by t: with $\underline{\boldsymbol{\theta}}_0 = \boldsymbol{\theta}_k$ and for $t = 0, \cdots, T-1$,

$$\mathbf{v}_t = \nabla f_{s_t}(\underline{\boldsymbol{\theta}}_t) - \nabla f_{s_t}(\underline{\boldsymbol{\theta}}_0) + \nabla F(\underline{\boldsymbol{\theta}}_0), \qquad \underline{\boldsymbol{\theta}}_{t+1} = \underline{\boldsymbol{\theta}}_t - \alpha \cdot \mathbf{v}_t, \tag{7}$$

where the index s_t is selected uniformly at random from $\{1, \cdots, N\}$. After the inner loop completes, $\boldsymbol{\theta}_{k+1}$ can be updated in a few different ways; applicable choices include setting $\boldsymbol{\theta}_{k+1}$

as $\underline{\boldsymbol{\theta}}_T$, $\frac{1}{T}\sum_{t=0}^{T-1}\underline{\boldsymbol{\theta}}_t$, or choosing it randomly from the inner loop updates $\{\underline{\boldsymbol{\theta}}_t\}_{t=0}^{T-1}$. For instance, assuming that each $f_i \in \mathcal{S}_{\mu,L}$, it can be shown that with $\boldsymbol{\theta}_{k+1} = \frac{1}{T}\sum_{t=0}^{T-1}\underline{\boldsymbol{\theta}}_t$, $\alpha = \frac{1}{10L}$, and $T = 50\kappa$, we have [8],

$$\mathbb{E}[\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2] \leq D \cdot 0.5^k, \qquad \forall k \geq 0,$$

for some $D > 0$. That is to say, SVRG achieves ϵ-accuracy with $\mathcal{O}(\log\frac{1}{\epsilon})$ outer-loop iterations. We further note that each outer-loop update requires $N + 2T$ component gradient evaluations (7). Therefore, SVRG achieves ϵ-accuracy of $\boldsymbol{\theta}^*$ with $\mathcal{O}\left((N+\kappa)\log\frac{1}{\epsilon}\right)$ component gradient evaluations, which is comparable to the convergence rate of SAGA.

Remark 1 (SGD with decaying step-sizes vs. VR): SGD, converging at a sublinear rate $\mathcal{O}(1/k)$ to the minimizer (4), typically makes a fast progress in its early stage for certain large-scale, complex machine learning tasks and then slows down considerably. Its complexity (4) is not explicitly dependent on the sample size N, which is a strong feature, but it comes at a price of a direct dependence on σ^2 (the variance of the stochastic gradient). On the other hand, the VR methods achieve fast linear convergence with the help of refined gradient estimators, for example, $\mathbf{g}_k$ or $\mathbf{v}_t$, which approach the corresponding batch gradients as their variance diminishes. Their convergence, although dependent on the sample size N, is independent of σ^2.

Remark 2 (SAGA vs. SVRG): The fundamental trade-off between SAGA and SVRG is convergence speed versus storage and is often described as a trade-off between time and space [7]. Although SAGA and SVRG in theory achieve convergence rates of the same order, SVRG in practice requires 2-3 times more component gradient evaluations to reach the same accuracy as SAGA, however, without storing all the component gradients [7].

In the rest of this article, we show how to cast SGD and VR methods in the decentralized optimization framework. *Section: Problem Formulation* describes the decentralized optimization problem over a network of nodes. In *Section: Decentralized Stochastic Optimization*, we extend centralized SGD to the decentralized problem and show that an appropriate decentralization is achieved with the help of gradient tracking. Subsequently, in *Section: Decentralized VR Methods*, we describe recent advances in decentralized methods that combine gradient tracking and variance reduction. *Section: Numerical Illustrations* provides numerical experiments on strongly-convex and non-convex problems and further highlights different tradeoffs between the methods described in this article. *Section: Extensions and Discussion* summarizes certain extensions and communication/computation aspects of the corresponding problems that are popular in the literature. Finally, *Section: Conclusions* concludes the paper and briefly describe some open problems.

II. Problem Formulation: Decentralized Empirical Risk Minimization

In this article, our focus is on the solutions for optimization problems that arise in peer-to-peer decentralized networks. Unlike traditional master-worker architectures, where a central node acts as a master that coordinates communication with all workers; there is no central coordinator in peer-to-peer networks and each node is only able to communicate with its immediate neighbors, see Fig. 2. The canonical form of decentralized optimization problems can be described as follows. Consider n nodes, such as machines, devices, or robots, that communicate over a static undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \cdots, n\}$ is the set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, i.e., a collection of ordered pairs $(i, r), i, r \in \mathcal{V}$, such that nodes i and r can exchange information. Following the discussion in *Section: Empirical Risk Minimization*, each node i holds a *local risk function*, $\widetilde{f}_i : \mathbb{R}^p \to \mathbb{R}$, not accessible by any other node in the network. The decentralized risk minimization problem can thus be defined as

$$\text{P2:} \quad \widetilde{\boldsymbol{\theta}}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^p} \widetilde{F}(\boldsymbol{\theta}), \qquad \widetilde{F}(\boldsymbol{\theta}) \triangleq \frac{1}{n} \sum_{i=1}^{n} \widetilde{f}_i(\boldsymbol{\theta}).$$

As in the centralized case with Problem P0, the underlying data distributions at the nodes may not be available or tractable, we thus employ a local empirical risk function at each node as a surrogate of the local risk. Specifically, we consider each node i as a computing resource that stores/collects a local batch of m_i training samples that are possibly private (not shared with other nodes) and the corresponding local empirical risk function is decomposed over the local data samples as $f_i \triangleq \frac{1}{m_i} \sum_{j=1}^{m_i} f_{i,j}$. The goal of the networked nodes is to *agree* on the *optimal* solution of the following *decentralized empirical risk minimization* problem:

$$\text{P3:} \quad \boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^p} F(\boldsymbol{\theta}), \qquad F(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\theta}) \triangleq \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{m_i} \sum_{j=1}^{m_i} f_{i,j}(\boldsymbol{\theta}) \right).$$

The rest of this article is dedicated to the solutions of Problem P3.

Fig. 1. (Left) A master-worker network. (Right) Decentralized optimization in peer-to-peer networks.

III. Decentralized Stochastic Optimization

We now consider decentralized solutions of Problem P3. At each node i given the current estimate $\boldsymbol{\theta}_k^i$ of $\boldsymbol{\theta}^*$ at iteration k, related algorithms typically involve the following steps:

(1) Sample one or more component gradients from $\{\nabla f_{i,j}(\boldsymbol{\theta}_k^i)\}_{j=1}^{m_i}$;

(2) Fuse information with the available neighbors;

(3) Compute $\boldsymbol{\theta}_{k+1}^i$ according to a specific optimization protocol.

Recall that each node in the network only communicates with a few nearby nodes and only has partial knowledge of the global objective, see Fig. 2 (right). Due to this limitation, an information propagation mechanism is required that disseminates local information over the entire network. Decentralized optimization thus has two key components: (i) *agreement or consensus*–all nodes must agree on the same state, i.e., $\boldsymbol{\theta}_k^i \to \boldsymbol{\theta}_{cons}, \forall i$; and, (ii) *optimality*–the agreement should be on the minimizer of the global objective F, i.e., $\boldsymbol{\theta}_{cons} = \boldsymbol{\theta}^*$. Average-consensus algorithms [10] are information fusion protocols that enable each node to appropriately combine the vectors received from its neighbors and to agree on the average of the initial states of the nodes. They thus naturally serve as basic building blocks in decentralized optimization, added to which are local gradient corrections that steer the agreement to the global minimizer.

To describe average-consensus, we first associate the undirected and connected graph $\mathcal{G}$ with a primitive, symmetric, and doubly-stochastic $n \times n$ weight matrix $W = \{w_{ir}\}$, such that $w_{ir} \neq 0$ for each $(i, r) \in \mathcal{E}$. Clearly, we have $W = W^\top$ and $W\mathbf{1}_n = \mathbf{1}_n$, where $\mathbf{1}_n$ is the column vector of n ones. There are various ways of constructing such weights in a decentralized manner. Popular choices include the Laplacian and Metropolis weights, see [11] for details. Average-consensus [10] is given as follows. Each node i starts with some vector $\boldsymbol{\theta}_0^i \in \mathbb{R}^p$ and updates its state according to $\boldsymbol{\theta}_{k+1}^i = \sum_{r \in \mathcal{N}_i} w_{ir}\boldsymbol{\theta}_k^r$, $\forall k \geq 0$. It can be written in a vector form as

$$\boldsymbol{\theta}_{k+1} = (W \otimes I_p)\boldsymbol{\theta}_k, \tag{8}$$

where $\boldsymbol{\theta}_k = [{\boldsymbol{\theta}_k^1}^\top, \cdots, {\boldsymbol{\theta}_k^n}^\top]^\top$. Since W is primitive and doubly-stochastic[2], from the Perron-Frobenius theorem [12], we have $\lim_{k\to\infty} W^k = \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top$ and $\lim_{k\to\infty} \boldsymbol{\theta}_k = (W \otimes I_p)^k\boldsymbol{\theta}_0 = (\mathbf{1}_n \otimes I_p)\overline{\boldsymbol{\theta}}_0$, where $\overline{\boldsymbol{\theta}}_0 \triangleq \frac{(\mathbf{1}_n^\top \otimes I_p)\boldsymbol{\theta}_0}{n}$, at a linear rate of λ^k, and $\lambda \in [0, 1)$ is the spectral radius of $(W - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top)$. That is to say, the protocol in (8) enables an agreement across the nodes on the average $\overline{\boldsymbol{\theta}}_0$ of their initial states, at a linear rate. With the agreement protocol in place, we next introduce decentralized gradient descent and its stochastic variant that build on top of average-consensus.

[2]In the rest of this article, $W = \{w_{ir}\}$ denotes a collection of doubly-stochastic weights and $\lambda \in [0, 1)$ is the spectral radius of $(W - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top)$.

A. Decentralized Stochastic Gradient Descent (DSGD)

Recall that our focus is to solve Problem P3 in a decentralized manner, when the nodes exchange information over an arbitrary undirected graph. A well-known solution to this problem is Decentralized Gradient Descent (DGD) [13], [14], described as follows. Each node i starts with an arbitrary $\boldsymbol{\theta}_0^i \in \mathbb{R}^p$ and performs the following update:

$$\boldsymbol{\theta}_{k+1}^i = \sum_{r \in \mathcal{N}_i} w_{ir} \boldsymbol{\theta}_k^r - \alpha_k \nabla f_i \left(\boldsymbol{\theta}_k^i \right), \qquad k \geq 0. \tag{9}$$

Indeed, at each node i, DGD adds a local gradient correction to average-consensus based on the local data batch, i.e., all $f_{i,j}$'s, and is the prototype of many decentralized optimization protocols. To understand the iterations of DGD, we write them in a vector form. Let $\boldsymbol{\theta}_k$ and $\nabla \mathbf{f}(\boldsymbol{\theta}_k)$ collect all local estimates and gradients, respectively, i.e., $\boldsymbol{\theta}_k = [{\boldsymbol{\theta}_k^1}^\top, \cdots, {\boldsymbol{\theta}_k^n}^\top]^\top$ and $\nabla \mathbf{f}(\boldsymbol{\theta}_k) \triangleq [\nabla f_1(\boldsymbol{\theta}_k^1)^\top, \cdots, \nabla f_n(\boldsymbol{\theta}_k^n)^\top]^\top$, both in $\mathbb{R}^{np}$. Then DGD can be compactly written as

$$\boldsymbol{\theta}_{k+1} = (W \otimes I_p)\boldsymbol{\theta}_k - \alpha_k \nabla \mathbf{f}(\boldsymbol{\theta}_k). \tag{10}$$

We further define the average $\overline{\boldsymbol{\theta}}_k \triangleq \frac{1}{n}(\mathbf{1}_n^\top \otimes I_p)\boldsymbol{\theta}_k$ of the local estimates at time k and multiply both sides of (10) by $\frac{1}{n}(\mathbf{1}_n^\top \otimes I_p)$ to obtain:

$$\overline{\boldsymbol{\theta}}_{k+1} = \overline{\boldsymbol{\theta}}_k - \alpha_k \frac{(\mathbf{1}_n^\top \otimes I_p)\nabla \mathbf{f}\left(\boldsymbol{\theta}_k\right)}{n}. \tag{11}$$

Based on (10) and (11), we note that the consensus matrix W makes the estimates $\{\boldsymbol{\theta}_k^i\}_{i=1}^n$ at the nodes approach their average $\overline{\boldsymbol{\theta}}_k$, while the average gradient $\frac{(\mathbf{1}_n^\top \otimes I_p)\nabla \mathbf{f}(\boldsymbol{\theta}_k)}{n}$ steers $\overline{\boldsymbol{\theta}}_k$ towards the minimizer $\boldsymbol{\theta}^*$ of F. The overall protocol thus ensures agreement and optimality, the two key components of decentralized optimization as we described before.

DGD is a simple yet effective method for various decentralized learning tasks. To make DGD efficient for large-scale decentralized empirical risk minimization, where each m_i is very large, Refs. [14], [15] derive a stochastic variant, known as *Decentralized Stochastic Gradient Descent (DSGD)*, by substituting each local batch gradient with a randomly sampled component gradient. DSGD is formally described in Algorithm 1. Assuming that each $f_{i,j} \in \mathcal{S}_{\mu,L}$ and each local stochastic gradient has bounded variance[3], i.e., $\mathbb{E}_{s_k^i}\left[\left\|\nabla f_{i,s_k^i}(\boldsymbol{\theta}_k^i) - \nabla f_i(\boldsymbol{\theta}_k^i)\right\|_2^2 | \boldsymbol{\theta}_k^i\right] \leq \sigma^2, \forall i, k$, we have [16]: under a constant step-size, $\alpha_k = \alpha \in \left(0, \mathcal{O}\left(\frac{(1-\lambda)}{L\kappa}\right)\right], \forall k$, $\mathbb{E}[\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\|_2^2]$ decays at a linear rate of $(1 - \mathcal{O}(\mu\alpha))^k$ to a neighborhood of $\boldsymbol{\theta}^*$ such that

$$\limsup_{k \to \infty} \frac{1}{n} \sum_{i=1}^n \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] = \mathcal{O}\left(\frac{\alpha\sigma^2}{n\mu} + \frac{\alpha^2\kappa^2\sigma^2}{1-\lambda} + \frac{\alpha^2\kappa^2 b}{(1-\lambda)^2}\right), \tag{12}$$

[3]The bounded variance assumption can also be relaxed as noted in Footnote 1 for the centralized case, see [16].

where $b \triangleq \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(\boldsymbol{\theta}^*)\|^2$ and $\kappa = L/\mu$. With a diminishing step-size $\alpha_k = \mathcal{O}(\frac{1}{k})$, DSGD achieves an exact convergence [17], [18], such that

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] = \mathcal{O}\left(\frac{1}{k}\right), \qquad \forall k \geq 0. \tag{13}$$

Algorithm 1 DSGD: At each node i

Require: $\boldsymbol{\theta}_0^i$, $\{\alpha_k\}_{k\geq 0}$, $\{w_{ir}\}_{r\in\mathcal{N}_i}$.

1: **for** $k = 0, 1, 2, \cdots$ **do**

2: **Choose** s_k^i uniformly at random in $\{1, \cdots, m_i\}$

3: **Compute** the local stochastic gradient $\nabla f_{i,s_k^i}(\boldsymbol{\theta}_k^i)$.

4: **Update**: $\boldsymbol{\theta}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\boldsymbol{\theta}_k^r - \alpha_k \nabla f_{i,s_k^i}(\boldsymbol{\theta}_k^i)$

5: **end for**

Remark 3 (SGD vs. DSGD): Comparing (3) to (12), when a constant step-size α is used, the steady-state error in both SGD and DSGD decays linearly to a certain neighborhood (controlled by α) of $\boldsymbol{\theta}^*$. Unlike SGD, however, the steady-state error of DSGD has an additional bias, independent of the variance σ^2 of the stochastic gradient, that comes from $b = \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(\boldsymbol{\theta}^*)\|^2$. The constant b is not zero in general and characterizes the difference between the minimizer of each local objective f_i and that of the global objective F. The resulting bias $\mathcal{O}\big(\frac{\alpha^2\kappa^2 b}{(1-\lambda)^2}\big)$ can be significantly large when the data distributions across nodes are substantially heterogeneous or when the graph is not well-connected, a scenario that commonly arises in certain wireless networks and IoT applications, see *Section: Numerical Illustrations*. In the following, we describe a gradient tracking technique that eliminates the bias in DSGD due to the term b and thus can be considered as a more appropriate decentralization of the centralized SGD.

B. Decentralized First-Order Methods with Gradient Tracking

To present the intuition behind the gradient tracking technique, we first recall the iterations of the (non-stochastic) Decentralized Gradient Descent (DGD) with a constant step-size in (9). Let us first assume, for the sake of argument, that all nodes agree on the minimizer of F at some iteration k, i.e., $\boldsymbol{\theta}_k^i = \boldsymbol{\theta}^*, \forall i$. Then at the next iteration $k+1$, we have

$$\boldsymbol{\theta}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\boldsymbol{\theta}^* - \alpha\nabla f_i(\boldsymbol{\theta}^*) = \boldsymbol{\theta}^* - \alpha\nabla f_i(\boldsymbol{\theta}^*), \tag{14}$$

where $\boldsymbol{\theta}^* - \nabla f_i(\boldsymbol{\theta}^*) \neq \boldsymbol{\theta}^*$, in general. In other words, the minimizer $\boldsymbol{\theta}^*$ is not necessarily a fixed point of (9). Of course, using the gradient $\nabla F\left(\boldsymbol{\theta}_k^i\right)$ of the *global* objective, instead of $\nabla f_i\left(\boldsymbol{\theta}_k^i\right)$, overcomes this issue but the global gradient is not available at any node. The natural yet innovative

idea of gradient tracking is to design a local iterative gradient tracker $\mathbf{d}_k^i$ that asymptotically approaches the global gradient $\nabla F\left(\boldsymbol{\theta}_k^i\right)$ as $\boldsymbol{\theta}_k^i$ approaches $\boldsymbol{\theta}^*$ [19]–[23]. Gradient tracking is implemented with the help of dynamic average consensus (DAC) [24], briefly described next.

In contrast to classical average-consensus [10] that learns the average of fixed initial states, DAC [24] tracks the average of time-varying signals. Formally, each node i measures a time-varying signal $\mathbf{r}_k^i$ and all nodes cooperate to track the average $\overline{\mathbf{r}}_k \triangleq \frac{1}{n}\sum_{i=1}^n \mathbf{r}_k^i$ of these signals. The DAC protocol is given as follows. Each node i iteratively updates its estimate $\mathbf{d}_k^i$ of $\overline{\mathbf{r}}_k$ as

$$\mathbf{d}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\mathbf{d}_k^r + \mathbf{r}_{k+1}^i - \mathbf{r}_k^i, \qquad k\geq 0, \tag{15}$$

where $\mathbf{d}_0^i = \mathbf{r}_0^i, \forall i$. For a doubly-stochastic weight matrix $W=\{w_{ir}\}$, it is shown in [24] that if $\left\|\mathbf{r}_{k+1}^i - \mathbf{r}_k^i\right\|_2 \to 0$, we have that $\left\|\mathbf{d}_k^i - \overline{\mathbf{r}}_k\right\|_2 \to 0$. Clearly, in the aforementioned design of gradient tracking, the time-varying signal that we intend to track is the average of the local gradients $\frac{1}{n}\sum_{i=1}^n \nabla f_i\left(\boldsymbol{\theta}_k^i\right)$. We thus combine DGD (9) and DAC (15) to obtain *GT-DGD (DGD with Gradient Tracking)* [19]–[23], as follows:

$$\boldsymbol{\theta}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\boldsymbol{\theta}_k^r - \alpha\cdot\mathbf{d}_k^i, \tag{16a}$$

$$\mathbf{d}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\mathbf{d}_k^r + \nabla f_i\left(\boldsymbol{\theta}_{k+1}^i\right) - \nabla f_i\left(\boldsymbol{\theta}_k^i\right), \tag{16b}$$

where $\mathbf{d}_0^i = \nabla f_i\left(\boldsymbol{\theta}_0^i\right), \forall i$. Intuitively, as $\boldsymbol{\theta}_k^i \to \overline{\boldsymbol{\theta}}_k$ and $\mathbf{d}_k^i \to \frac{1}{n}\sum_{i=1}^n \nabla f_i\big(\boldsymbol{\theta}_k^i\big) \to \nabla F\big(\overline{\boldsymbol{\theta}}_k\big)$, (16a) asymptotically becomes the centralized batch gradient descent. It has been shown in [21]–[23], [25] that GT-DGD converges linearly to the minimizer $\boldsymbol{\theta}^*$ of F under a *constant step-size* when each $f_{i,j}\in\mathcal{S}_{\mu,L}$, unlike DGD that converges sublinearly to $\boldsymbol{\theta}^*$ with decaying step-sizes.

The stochastic variant of GT-DGD is derived in [26], termed as *GT-DSGD (DSGD with Gradient Tracking)*, and is formally described in Algorithm 2. Under the same assumptions of smoothness, strong-convexity, and bounded variance as in DSGD, the convergence of GT-DSGD is summarized in the following [26]: with a constant step-size, $\alpha_k = \alpha \in \left(0, \mathcal{O}\left(\frac{(1-\lambda)^2}{L\kappa}\right)\right], \forall k$, $\mathbb{E}[\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\|_2^2]$ decays linearly at the rate of $(1-\mathcal{O}(\mu\alpha))^k$ to a neighborhood of $\boldsymbol{\theta}^*$ such that

$$\limsup_{k\to\infty}\frac{1}{n}\sum_{i=1}^n \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] = \mathcal{O}\left(\frac{\alpha\sigma^2}{n\mu} + \frac{\alpha^2\sigma^2\kappa^2}{(1-\lambda)^3}\right). \tag{17}$$

Note that GT-DSGD, in contrast to GT-DGD, loses the exact linear convergence to the minimizer because the gradients are now stochastic. Exact convergence can be recovered albeit at a slower sublinear rate, i.e., with a diminishing step-size $\alpha_k = \mathcal{O}(\frac{1}{k})$, we have [26]

$$\frac{1}{n}\sum_{i=1}^n \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] = \mathcal{O}\left(\frac{1}{k}\right), \qquad \forall k\geq 0. \tag{18}$$

Algorithm 2 GT-DSGD: At each node i

Require: $\boldsymbol{\theta}_0^i$, $\{\alpha_k\}_{k\geq 0}$, $\{w_{ir}\}_{r\in\mathcal{N}_i}$, $\mathbf{d}_0^i = \nabla f_{i,s_0^i}(\boldsymbol{\theta}_0^i)$, where s_0^i is chosen uniformly at random in $\{1,\cdots,m_i\}$

1: **for** $k = 0, 1, 2, \cdots$ **do**

2: **Update** $\boldsymbol{\theta}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\boldsymbol{\theta}_k^r - \alpha_k \mathbf{d}_k^i$

3: **Choose** s_{k+1}^i uniformly at random in $\{1,\cdots,m_i\}$

4: **Compute** the local stochastic gradient $\nabla f_{i,s_{k+1}^i}(\boldsymbol{\theta}_{k+1}^i)$

5: **Update**: $\mathbf{d}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\mathbf{d}_k^r + \nabla f_{i,s_{k+1}^i}(\boldsymbol{\theta}_{k+1}^i) - \nabla f_{i,s_k^i}(\boldsymbol{\theta}_k^i)$

6: **end for**

Remark 4 (DSGD vs. GT-DSGD): By comparing DSGD (12) and GT-DSGD (17), we note that under a constant step-size, GT-DSGD removes the bias $\mathcal{O}\left(\frac{\alpha^2\kappa^2 b}{(1-\lambda)^2}\right)$ that comes from $b \triangleq \frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\boldsymbol{\theta}^*)\|^2$ in DSGD. However, the network dependence in GT-DSGD, $\mathcal{O}\left(\frac{1}{(1-\lambda)^3}\right)$, is worse than DSGD where it is $\mathcal{O}\left(\frac{1}{(1-\lambda)^2}\right)$. A tradeoff here is imminent where the two approaches have their own merits depending on the relative sizes of b and λ. Clearly, when the bias b dominates, e.g., when the data across nodes is largely heterogeneous, GT-DSGD achieves a lower steady-state error than DSGD. Under diminishing step-sizes, DSGD and GT-DSGD have comparable performance. Of relevance here are EXTRA [27] and Exact Diffusion [28], both of which eliminate the bias caused by b and are built on a different principle from gradient tracking.

Remark 5 (SGD vs. GT-DSGD): Note that with constant step-sizes, the performance of SGD in (3) and GT-DSGD in (17) is comparable. In particular, both methods converge linearly but there is a steady-state error, which is controlled by the step-size α and the variance σ^2 of the stochastic gradient, see Remark 3. Since GT-DSGD removes the bias in DSGD that comes due to the difference of the local and global objectives (see b in (12)), it may be considered as a more appropriate decentralization of SGD. This argument naturally leads to the idea that one can incorporate the centralized Variance Reduction (VR) techniques in the GT-DSGD to further improve the performance and achieve faster convergence. As we show in the following, adding variance reduction to GT-DSGD in fact leads to an exact linear convergence with a *constant* step-size and further improves its network dependence to $\mathcal{O}\left(\frac{1}{(1-\lambda)^2}\right)$.

Remark 6 (DSGD + VR): We emphasize that adding VR to DSGD does not enable exact linear convergence. Following Remark 1, VR removes the steady-state error caused by the variance of the stochastic gradient. However, in a decentralized setting, the heterogeneity across the local data

batches is not accounted for unless gradient tracking is employed. This difference between the local batches across the nodes is captured by the aforementioned bias b in (12) and is removed by gradient tracking that estimates the average of local gradients across the nodes.

IV. Decentralized Variance-Reduced Methods with Gradient Tracking

We now provide a unified algorithmic framework, GT-VR, that provably improves DSGD and follows from Remarks 5 and 6. This framework combines variance-reduction with GT-DSGD to achieve both robust performance and fast convergence. First, recall from *Section: Variance-Reduced Stochastic Gradient Descent* that VR methods iteratively estimate the batch gradient from randomly drawn samples. In the decentralized case, each node i thus implements VR locally to estimate its local batch gradient ∇f_i. Gradient tracking, on the other hand, estimates the average of the local VR estimators across the nodes and can be thought of as fusion in space. Consequently, VR and gradient tracking jointly learn the global batch gradient ∇F at each node asymptotically. For definiteness, we present and analyze two instances of GT-VR, namely, GT-SAGA and GT-SVRG, and show that they achieve exact linear convergence with constant step-sizes for the class of smooth and strongly-convex functions. We further show that in a "big-data" regime, both GT-SAGA and GT-SVRG act effectively as means for parallel computation and achieve a linear speed-up compared with their centralized counterparts.

A. GT-SAGA

To implement the SAGA estimators locally, each node i maintains a gradient table that stores all local component gradients $\{\nabla f_{i,j}(\widehat{\boldsymbol{\theta}}_{i,j})\}_{j=1}^{m_i}$, where $\widehat{\boldsymbol{\theta}}_{i,j}$ represents the most recent iterate where the gradient of $f_{i,j}$ was evaluated. At iteration $k \geq 0$, each node i chooses an index s_k^i uniformly at random from $\{1, \cdots, m_i\}$ and computes the local SAGA gradient $\mathbf{g}_k^i$ as

$$\mathbf{g}_k^i = \nabla f_{i,s_k^i}\big(\boldsymbol{\theta}_k^i\big) - \nabla f_{i,s_k^i}\big(\widehat{\boldsymbol{\theta}}_{i,s_k^i}\big) + \frac{1}{m_i}\sum_{j=1}^{m_i} \nabla f_{i,j}\big(\widehat{\boldsymbol{\theta}}_{i,j}\big), \tag{19}$$

where it can be shown that $\mathbf{g}_k^i$ is an unbiased estimator of the local batch gradient $\nabla f_i(\boldsymbol{\theta}_k^i)$. Next, the element $\nabla f_{i,s_k^i}(\widehat{\boldsymbol{\theta}}_{i,s_k^i})$ in the gradient table is replaced by $\nabla f_{i,s_k^i}\big(\boldsymbol{\theta}_k^i\big)$, while the other elements remain unchanged. The gradient tracking iteration $\mathbf{d}_k^i$ is then implemented on the estimators $\mathbf{g}_k^i$'s. The complete implementation of GT-SAGA [29] is summarized in Algorithm 3.

Similar to centralized SAGA [7], GT-SAGA converges linearly to $\boldsymbol{\theta}^*$ with a constant step-size. More precisely, assuming each $f_{i,j} \in \mathcal{S}_{\mu,L}$ and by choosing $\alpha = \min\left\{\mathcal{O}\left(\frac{1}{\mu M}\right), \mathcal{O}\left(\frac{m}{M}\frac{(1-\lambda)^2}{L\kappa}\right)\right\}$,

Algorithm 3 GT-SAGA at each node i

Require: $\boldsymbol{\theta}_0^i$, α, $\{w_{ir}\}_{r\in\mathcal{N}_i}$, $\mathbf{d}_0^i = \mathbf{g}_0^i = \nabla f_i(\boldsymbol{\theta}_0^i)$, Gradient table $\{\nabla f_{i,j}(\widehat{\boldsymbol{\theta}}_{i,j})\}_{j=1}^{m_i}$, $\widehat{\boldsymbol{\theta}}_{i,j} = \boldsymbol{\theta}_0^i, \forall j$.

1: **for** $k = 0, 1, 2, \cdots$ **do**

2: **Update** $\boldsymbol{\theta}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\boldsymbol{\theta}_k^r - \alpha\mathbf{d}_k^i$;

3: **Choose** s_{k+1}^i uniformly at random from $\{1, \cdots, m_i\}$;

4: **Compute** $\mathbf{g}_{k+1}^i = \nabla f_{i,s_{k+1}^i}(\boldsymbol{\theta}_{k+1}^i) - \nabla f_{i,s_{k+1}^i}(\widehat{\boldsymbol{\theta}}_{i,s_{k+1}^i}) + \frac{1}{m_i}\sum_{j=1}^{m_i}\nabla f_{i,j}(\widehat{\boldsymbol{\theta}}_{i,j})$;

5: **Replace** $\nabla f_{i,s_{k+1}^i}(\widehat{\boldsymbol{\theta}}_{i,s_{k+1}^i})$ by $\nabla f_{i,s_{k+1}^i}(\boldsymbol{\theta}_{k+1}^i)$ in the gradient table.

6: **Update** $\mathbf{d}_{k+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\mathbf{d}_k^r + \mathbf{g}_{k+1}^i - \mathbf{g}_k^i$;

7: **end for**

where $m = \min_i\{m_i\}, M = \max_i\{m_i\}$, we have [29],

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] \leq R\left(1 - \min\left\{\mathcal{O}\left(\frac{1}{M}\right), \mathcal{O}\left(\frac{m}{M}\frac{(1-\lambda)^2}{\kappa^2}\right)\right\}\right)^k, \qquad \forall k \geq 0, \quad (20)$$

for some $R > 0$. In other words, GT-SAGA achieves ϵ-accuracy of $\boldsymbol{\theta}^*$ in

$$\mathcal{O}\left(\max\left\{M, \frac{M}{m}\frac{\kappa^2}{(1-\lambda)^2}\right\}\log\frac{1}{\epsilon}\right)$$

parallel local component gradient computations. We emphasize that GT-SAGA, unlike the stochastic algorithms (DSGD and GT-DSGD) discussed before, exhibits linear convergence to the global minimizer $\boldsymbol{\theta}^*$ of F. This exact linear convergence is a consequence of both variance reduction and gradient tracking; see Remarks 7, 8, 9 and 10 for additional comments.

B. GT-SVRG

GT-SVRG, formally described in Algorithm 4, is a double-loop method, where the outer loop index is k and the inner loop index is t, that builds upon the centralized SVRG. At every outer loop, each node i computes a local batch gradient and proceeds to a finite number T of inner loop iterations; in the inner loop, each node i performs GT-DSGD (type) iterations in addition to updating the local gradient estimate $\mathbf{v}_t^i$ (Algorithm 4: Step 7). It can be verified that $\mathbf{v}_t^i$ is an unbiased estimator of the corresponding local batch gradient at node i. In practice, all options (a)-(c) work similarly well. For example, under option (a), it is shown in [29] that with $\alpha = \mathcal{O}\left(\frac{(1-\lambda)^2}{L\kappa}\right)$ and $T = \mathcal{O}\left(\frac{\kappa^2\log\kappa}{(1-\lambda)^2}\right)$, the outer loop of GT-SVRG follows:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\left\|\boldsymbol{\theta}_k^i - \boldsymbol{\theta}^*\right\|_2^2\right] \leq U\cdot 0.9^k, \quad (21)$$

for some $U > 0$. This argument implies that GT-SVRG achieves ϵ-accuracy of $\boldsymbol{\theta}^*$ in $\mathcal{O}\left(\log\frac{1}{\epsilon}\right)$ outer loop iterations. We further note that each outer-loop update requires each node i to com-

pute $m_i + 2T$ local component gradients. GT-SVRG thus achieves ϵ-accuracy of $\boldsymbol{\theta}^*$ in totally

$$\mathcal{O}\left(\left(M + \frac{\kappa^2 \log \kappa}{(1-\lambda)^2}\right) \log \frac{1}{\epsilon}\right)$$

parallel local component gradient computations.

Algorithm 4 GT-SVRG at each node i

Require: $\boldsymbol{\theta}_0^i$, α, $\{w_{ir}\}_{r\in\mathcal{N}_i}$, $\mathbf{d}_0^i = \mathbf{v}_0^i = \nabla f_i(\boldsymbol{\theta}_0^i)$.

for $k = 0, 1, 2, \cdots$ **do**
  Initialize $\underline{\boldsymbol{\theta}}_0^i = \boldsymbol{\theta}_k^i$
  Compute $\nabla f_i(\underline{\boldsymbol{\theta}}_0^i) = \frac{1}{m_i}\sum_{j=1}^{m_i} \nabla f_{i,j}(\underline{\boldsymbol{\theta}}_0^i)$
  for $t = 0, 1, 2, \cdots, T-1$ **do**
    Update $\underline{\boldsymbol{\theta}}_{t+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\underline{\boldsymbol{\theta}}_t^r - \alpha \cdot \mathbf{d}_t^i$;
    Choose s_{t+1}^i uniformly at random from $\{1, \cdots, m_i\}$;
    Compute $\mathbf{v}_{t+1}^i = \nabla f_{i,s_{t+1}^i}(\underline{\boldsymbol{\theta}}_{t+1}^i) - \nabla f_{i,s_{t+1}^i}(\underline{\boldsymbol{\theta}}_0^i) + \nabla f_i(\underline{\boldsymbol{\theta}}_0^i)$;
    Update $\mathbf{d}_{t+1}^i = \sum_{r\in\mathcal{N}_i} w_{ir}\mathbf{d}_t^r + \mathbf{v}_{t+1}^i - \mathbf{v}_t^i$;
  end for
  Set $\mathbf{d}_0^i = \mathbf{d}_T^i$ and $\mathbf{v}_0^i = \mathbf{v}_T^i$
  Option (a): **Set** $\boldsymbol{\theta}_{k+1}^i = \underline{\boldsymbol{\theta}}_T^i$
  Option (b): **Set** $\boldsymbol{\theta}_{k+1}^i = \frac{1}{T}\sum_{t=0}^{T-1} \underline{\boldsymbol{\theta}}_t^i$
  Option (c): **Set** $\boldsymbol{\theta}_{k+1}^i$ as a random selection from $\{\underline{\boldsymbol{\theta}}_t^i\}_{t=0}^{T-1}$
end for

Remark 7 (GT-SAGA vs. GT-SVRG: Linear speedup): Both GT-SAGA and GT-SVRG have a low per-iteration computation cost and converge linearly to $\boldsymbol{\theta}^*$, i.e., they reach ϵ-accuracy of $\boldsymbol{\theta}^*$ respectively in $\mathcal{O}\left(\max\left\{M, \frac{M}{m}\frac{\kappa^2}{(1-\lambda)^2}\right\}\log\frac{1}{\epsilon}\right)$ and $\mathcal{O}\left(\left(M + \frac{\kappa^2\log\kappa}{(1-\lambda)^2}\right)\log\frac{1}{\epsilon}\right)$ parallel local component gradient computations. Interestingly, when the data sets at the nodes are large and balanced such that $M \approx m \gg \frac{\kappa^2}{1-\lambda^2}$, the complexities of GT-SAGA and GT-SVRG become $\mathcal{O}(M\log\frac{1}{\epsilon})$, independent of the network, and are n times faster than that of centralized SAGA and SVRG. Clearly, in this "big-data" regime, GT-SAGA and GT-SVRG each acts effectively as a means for parallel computation and achieves a linear speed-up compared with its centralized counterpart.

Remark 8 (GT-SAGA vs. GT-SVRG: Unbalanced data): It can also be observed that when data samples are distributed over the network in an unbalanced way, i.e., $\frac{M}{m}$ is large, GT-SVRG may achieve a lower complexity than GT-SAGA in terms of number of component gradient evaluations. However, from a practical implementation standpoint, an unbalanced data distribution may lead to a longer wall-clock time in GT-SVRG. This is because the next inner loop cannot be executed

until all nodes finish their local batch gradient computations and nodes with a large amount of data take longer to finish this computation, leading to an overall increase in runtime. Clearly, there is an inherent trade-off between network synchrony, latency, and the storage of gradients as far as the relative implementation complexities of GT-SAGA and GT-SVRG are concerned. If each node is capable of storing all local component gradients, then GT-SAGA is preferable due to its flexibility of implementation and faster convergence in practice. On the other hand, for large-scale optimization problems where each node holds a very large number of data samples, storing all component gradients may be infeasible and therefore GT-SVRG may be preferred.

Remark 9 (Related work on decentralized VR methods): Existing decentralized VR methods include DSA [30] that combines EXTRA [27] with SAGA [7], diffusion-AVRG that combines exact diffusion [28] and AVRG [31], DSBA [32] that adds proximal mapping [33] to each iteration of DSA, ADFS [34] that applies an accelerated randomized proximal coordinate gradient method [35] to the dual formulation of Problem P3, and Network-SVRG/SARAH [36] that implements variance-reduction in the decentralized DANE framework based on gradient tracking. We note that in large-scale scenarios where $M \approx m$ is very large, both GT-SAGA and GT-SVRG improve upon the convergence rate of these methods in terms of the joint dependence on κ and $M \approx m$, with the exception of DSBA and ADFS. Both DSBA and ADFS achieve better iteration complexity, however, at the expense of computing the proximal mapping of a component function at each iteration. Although the computation of this proximal mapping is efficient for certain function classes, it can be very expensive for general functions.

Remark 10 (Communication complexity): We now compare the communication complexities of the decentralized algorithms discussed in this article. Since the node deployment is not necessarily deterministic, we provide the expected number of communication rounds per node required to achieve an ϵ-accurate solution (each communication is over a p-dimensional vector). Note that DSGD, GT-DSGD, and GT-SAGA all incur $\mathcal{O}(d_{\text{exp}})$ expected number of communication rounds per node, at each iteration, where d_{exp} is the expected degree of the (possibly random) communication graph $\mathcal{G}$. Thus, their *expected communication complexity* is their iteration complexity scaled by d_{exp} and is given by $\mathcal{O}(d_{\text{exp}}\frac{1}{\epsilon})$, $\mathcal{O}(d_{\text{exp}}\frac{1}{\epsilon})$, and $\mathcal{O}\left(\max\left\{M, \frac{M}{m}\frac{\kappa^2}{(1-\lambda)^2}\right\} d_{\text{exp}} \log\frac{1}{\epsilon}\right)$, respectively. For GT-SVRG, we note that a total number of $\mathcal{O}(\log\frac{1}{\epsilon})$ outer-loop iterations are required, where each corresponding inner loop incurs $\mathcal{O}(T) = \mathcal{O}\left(\frac{\kappa^2\log\kappa}{(1-\lambda)^2} d_{\text{exp}}\right)$ rounds of communication, resulting into to a total communication complexity of $\mathcal{O}\left(\frac{\kappa^2\log\kappa}{(1-\lambda)^2} d_{\text{exp}} \log\frac{1}{\epsilon}\right)$. Clearly, GT-SAGA and GT-SVRG, due to their fast linear convergence, improve upon the communication

complexities of DSGD and GT-DSGD. It is further interesting to observe that in the big-data regime where each node has a large number of data samples, GT-SVRG achieves a lower communication complexity than GT-SAGA. Finally, we note that all gradient-tracking based algorithms require two consecutive rounds of communication per stochastic gradient evaluation with neighboring nodes to update the estimate $\boldsymbol{\theta}_k^i$ and the gradient tracker $\mathbf{d}_k^i$, respectively. This may increase the communication burden of the network especially when $\boldsymbol{\theta}_k^i$ is of high dimension. We note that, for the sake of completeness, we add d_{exp} to the communication complexities, which is a function of the underlying graph $\mathcal{G}$; in particular, $d_{\text{exp}} = \mathcal{O}(1)$ for random geometric graphs (assuming constant density of deployment of nodes) and $d_{\text{exp}} = \mathcal{O}(\log n)$ for exponential graphs, see also *Section: Numerical Illustrations* on these graphs.

V. Numerical Illustrations

In this section, we present numerical experiments to illustrate the convergence properties of the decentralized stochastic optimization algorithms discussed in this article, i.e., DSGD, GT-DSGD, GT-SAGA, and GT-SVRG. We show experimental results on two different types of graphs shown in Fig. 2: (i) an exponential graph with $n = 16$ nodes modeling a highly-structured training environment with a large number of data samples per node; and, (ii) a random geometric graph with $n = 1,000$ nodes modeling a large-scale, ad-hoc training scenario. Their associated doubly-stochastic weight matrices W are generated by the Metroplis method with the second largest eigenvalue λ of 0.75 in the former and 0.9994 in the latter. The decentralized training problem we consider is classification of hand-written digits from the MNIST dataset [37] with the help of logistic regression (strongly-convex) and a two-layer neural network (non-convex).

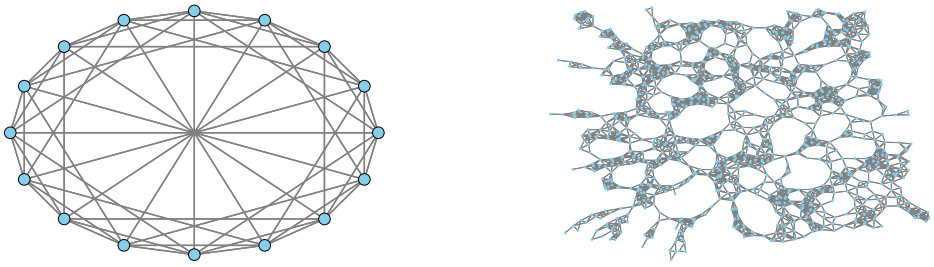

Fig. 2. (Left) An exponential graph with 16 nodes. (Right) A random geometric graph with $1,000$ nodes.

A. Logistic Regression: Strongly-convex

We first compare the algorithms of interest in the context of training a regularized logistic regression model [2], that is smooth and strongly-convex, to classify two digits $\{3, 8\}$. We use a total of $N = 12,000$ images for training and $1,966$ images for testing. Each node i holds m_i

training samples, i.e., $\{\mathbf{x}_{i,j}, y_{i,j}\}_{j=1}^{m_i} \subseteq \mathbb{R}^{784} \times \{-1, +1\}$, where $\mathbf{x}_{i,j}$ is the feature vector (image) and $y_{i,j}$ is the corresponding binary label. The nodes cooperate to solve the following problem:

$$\min_{\mathbf{b}\in\mathbb{R}^{784},\, c\in\mathbb{R}} F(\mathbf{b}, c) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{m_i}\sum_{j=1}^{m_i} \ln\left[1 + \exp\left\{-(\mathbf{b}^\top \mathbf{x}_{i,j} + c) y_{i,j}\right\}\right] + \frac{\lambda}{2}\|\mathbf{b}\|_2^2,$$

where $\boldsymbol{\theta} = [\mathbf{b}^\top c]^\top$, the regularization parameter is $\lambda = 1/N$, and the features are normalized to unit vectors [6], [38]. We plot the optimality gap, i.e., $F(\overline{\boldsymbol{\theta}}_k) - F(\boldsymbol{\theta}^*)$, vs. the number of parallel component gradient evaluations and compare the algorithms in both balanced and unbalanced data distribution scenarios, recall Remarks 7 and 8. The step-size for all algorithms is constant and is chosen to be $1/L$, while the inner-loop length T of GT-SVRG is N/n in the case of balanced data and $4N/n$ in the case of unbalanced data.

Balanced Data: To model a stable training environment with a balanced data distribution, e.g., in data centers or computing clusters, we choose a highly structured, well-connected, exponential graph with $n = 16$ nodes resulting into a relatively large number of samples ($m_i = 750$) per node. Each node has approximately the same number of images in each class, i.e., the data distribution is balanced and homogeneous, leading to similar local cost functions among the nodes and therefore the bias term b in DSGD is relatively small. From Remarks 3 and 4, recall that when b is small and the graph is well-connected, DSGD and GT-DSGD exhibit similar performance that is also verified numerically in Fig. 3. Adding variance reduction to GT-DSGD however significantly improves the performance in terms of both the optimality gap and the test accuracy, leading to a linear convergence in both GT-SAGA and GT-SVRG to the exact solution.

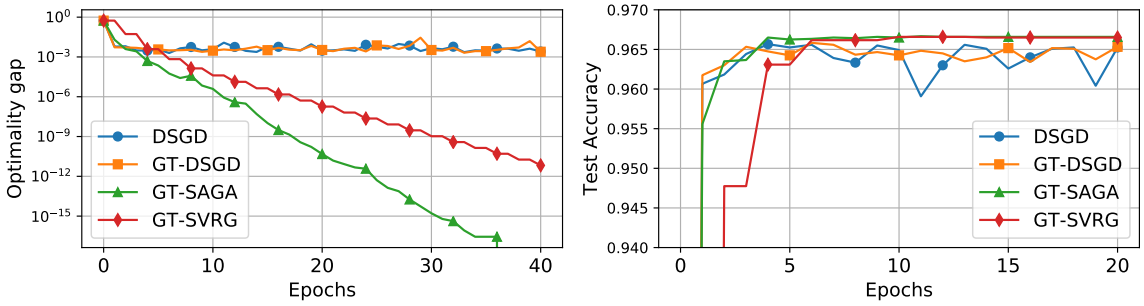

Fig. 3. Decentralized logistic regression with balanced data over the 16-node exponential graph, where each epoch represents $N/n = 750$ component gradient evaluations at each node.

Unbalanced Data: We next compare the algorithms when the data distribution is unbalanced and the nodes interact over a random geometric graph of $n = 1,000$ nodes, modeling a large-scale, wireless communication network. In this case, the $N = 12,000$ training images are randomly distributed among the nodes, see Fig. 4 (right) for the number of training samples at each node. We make a further restriction that the training data samples at each node belong to only one

class, either 3 or 8. This leads to unbalanced data sizes and heterogeneous data distributions at the nodes, making the local functions significantly different from each other and thus the bias b in DSGD is relatively large. The performance comparison is shown in Fig. 4 (left), where it can be observed that DSGD degrades considerably in this case and the addition of gradient tracking results into a smaller steady-state error (Remark 4). Adding variance reduction, as before, leads to a linear convergence to the exact solution.

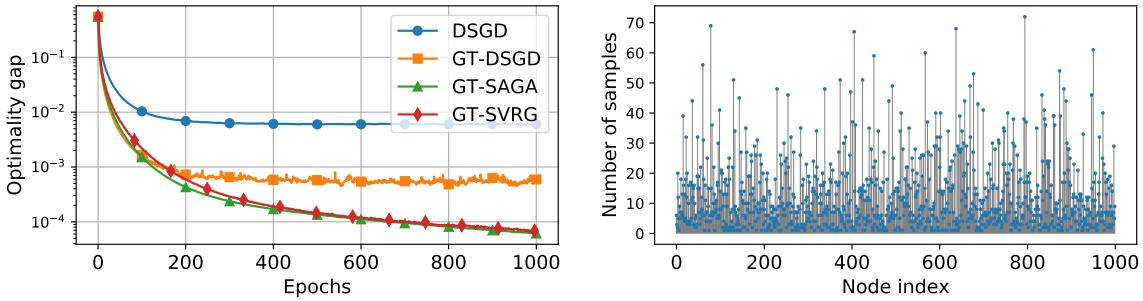

Fig. 4. Decentralized logistic regression with unbalanced data over a $1,000$-node random geometric graph, where each epoch represents $N/n = 12$ component gradient evaluations at each node.

Discussion: In both balanced and unbalanced data scenarios, the performance improvement due to gradient tracking comes at a price of one additional round of communication per iteration, see also Remark 10. The addition of variance reduction in GT-SAGA and GT-SVRG significantly outperforms both DSGD and GT-DSGD. Their linear convergence however comes at a price of additional storage in GT-SAGA and a synchronization overhead in GT-SVRG. From Remark 7, we recall that when each node has roughly the same number of training samples, GT-SAGA converges faster than GT-SVRG in terms of the number of parallel component gradient computations required, as can be observed in Fig. 3. On the other hand, as discussed in Remark 8, the iteration complexity of GT-SVRG is more robust to unbalanced data as it is independent of the M/m factor that appears in GT-SAGA, as it is shown in Fig. 4, where GT-SAGA and GT-SVRG exhibit similar convergence. However, GT-SVRG may incur additional latency and synchronization when the data is unbalanced, due to the different computing time of the local batch gradient evaluations across the network, before the execution of each inner-loop.

B. Neural Network: Non-convex

We now compare the performance of the algorithms when training a neural network with a non-convex loss function. The local neural network implemented at each node has one fully-connected hidden layer with 64 neurons and $51,675$ parameters in total. The goal is to train a neural network that classifies all ten digits $\{0,\dots,9\}$ from the MNIST dataset with $60,000$

training samples (around $6,000$ images in each class) and $10,000$ test images. The training dataset is divided randomly over $1,000$ nodes such that each node has 60 data points. All algorithms use a constant step-size that is manually optimized for best performance. Fig. 5 shows the loss $F(\overline{\boldsymbol{\theta}}_k)$ and the test accuracy over epochs. We note that adding gradient tracking to DSGD improves both the transient and steady-state performance in this non-convex setting. Similarly, adding variance-reduction improves the performance further. This behavior is also notable in the test accuracy.

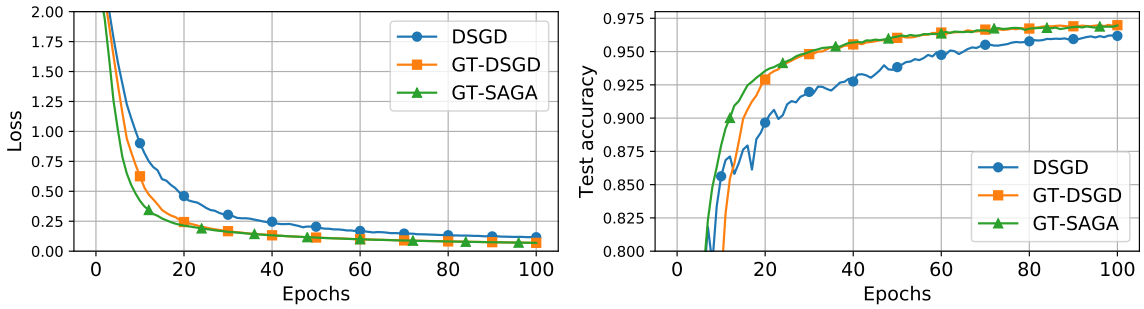

Fig. 5. Two layer neural network over a $1,000$-node random geometric graph, where one epoch represents $N/n = 60$ component gradient evaluations at each node.

VI. Extensions and Discussion

We now discuss some recent progress on several key aspects of decentralized optimization relevant to the first-order stochastic approaches described in this article.

Directed Graphs: The methods described in this article are restricted to undirected graphs. Over *directed graphs*, the main challenge is that the weight matrices are either row-stochastic (RS) or column-stochastic (CS), but cannot be doubly-stochastic (DS), in general. A well-studied solution to this issue is based on the push-sum (type) algorithms [39] that enable consensus with non-DS weights with the help of eigenvector estimation. Combining push-sum respectively with DSGD [14], [15], and GT-DGD [20]–[22] leads to SGP [40], and ADD-OPT [41] that require CS weights. A similar idea is used in FROST [42] to implement decentralized optimization with RS weights. The issue with push-sum based extensions is that they require eigenvector estimation, which in itself is an iterative procedure and may slow down the underlying algorithms especially when the corresponding communication graphs are not well-connected. More recently, it is shown that GT-DGD (16), ADD-OPT, and FROST are special cases of the AB algorithm [23], [43] that employs RS weights in (16a) and CS weights in (16b), and thus is immediately applicable to arbitrary directed graphs. The AB framework naturally leads to stochastic optimization with gradient tracking over directed graphs, see SAB [44] that extends GT-DSGD to directed graphs, and further opens the possibility to extend GT-SAGA and GT-SVRG to their directed counterparts.

Communication and computation aspects: Communication efficiency is an important aspect of decentralized optimization since communication can potentially become a bottleneck of the system when nodes are frequently transmitting high-dimensional vectors (model parameters) in the network. Different communication-efficient schemes [36], [45], communication/computation tradeoffs [11], asynchronous implementations [46], and quantization techniques [47], [48] have been studied with existing decentralized methods to efficiently manage the resources at each node.

Master-worker architectures: The problems described in this article have experienced a significant research activity because of their direct applicability to large-scale training problems in machine learning [40], [49]. Since these applications are typically hosted in controlled settings, e.g., data centers with highly-sophisticated communication and a large number of highly-efficient computing clusters, master-worker architectures and parameter-server models have become popular. In such architectures, see Fig. 2 (left), a central master maintains the current model parameters and communicates strategically with the workers, which individually hold a local batch of the total training data. Indeed, this architecture is not restricted to data centers alone and is also applicable to certain Internet-of-Things (IoT) scenarios where the devices are able to communicate to the master either directly via the cloud or via a mesh network among the devices. Various programming models and several variants of master-worker configurations have been proposed, such as MapReduce, All-Reduce, and federated learning [50], that are tailored for specific computing needs and environments. We emphasize that, on the contrary, the motivation behind the decentralized methods studied in this article comes from the scenarios where communication among the nodes is ad hoc, unstructured, and specialized topologies are not available.

VII. Conclusions

In this article, we discuss general formulation and solutions for decentralized, stochastic, first-order optimization methods. Our focus is on peer-to-peer networks that is applicable to ad-hoc wireless communication where the nodes have resource-constraints and limited communication capabilities. We discuss several fundamental algorithmic frameworks with a focus on gradient tracking and variance-reduction. For all algorithms, we provide a detailed discussion on their convergence rates, properties, and tradeoffs, with a particular emphasis on smooth and strongly-convex objective functions. An important line of future work in the field of decentralized machine learning is to analyze existing methods and develop new techniques for general non-convex objectives, given the tremendous success of deep neural networks.

SHORT BIOGRAPHIES

Ran Xin (ranx@andrew.cmu.edu) is a PhD candidate in the Electrical and Computer Engineering (ECE) department at Carnegie Mellon University (CMU), PA. His research interests include optimization theory and methods.

Soummya Kar (soummyak@andrew.cmu.edu) is an Associate Professor of ECE at Carnegie Mellon University, PA. His research interests include large-scale stochastic systems.

Usman A. Khan (khan@ece.tufts.edu) is an Associate Professor of ECE at Tufts University, MA. His research interests include optimization, control, and signal processing.

ACKNOWLEDGMENTS

The authors acknowledge the support of NSF under awards CCF-1350264, CCF-1513936, CMMI-1903972, and CBET-1935555. The authors would like to thank Boyue Li (CMU), Jianyu Wang (CMU), and Shuhua Yu (CMU) for their help and valuable discussions.

REFERENCES

[1] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[3] Y. Nesterov, *Lectures on convex optimization*, vol. 137, Springer, 2018.

[4] D. Needell, R. Ward, and N. Srebro, "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm," in *Advances in Neural Information Processing Systems*, pp. 1017–1025. 2014.

[5] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtarik, "SGD: General analysis and improved rates," in *International Conference on Machine Learning*, 2019.

[6] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[7] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in NIPS*, 2014, pp. 1646–1654.

[8] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[9] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, "SARAH: A novel method for machine learning problems using stochastic recursive gradient," in *34th International Conference on Machine Learning*, 2017, pp. 2613–2621.

[10] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[11] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 953–976, 2018.

[12] R. A. Horn and C. R. Johnson, *Matrix analysis*, Cambridge University Press, 2012.

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

[14] J. Chen and A. H. Sayed, "Diffusion adaptation strategies for distributed optimization and learning over networks," *IEEE Trans. Signal Process.*, vol. 60, no. 8, pp. 4289–4305, 2012.

[15] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *Journal of Optimization Theory and Applications*, vol. 147, no. 3, pp. 516–545, 2010.

[16] K. Yuan, S. A. Alghunaim, B. Ying, and A. H. Sayed, "On the performance of exact diffusion over adaptive networks," *arXiv:1903.10956*, 2019.

[17] D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar, "Convergence rates for distributed stochastic optimization over random networks," in *IEEE Conference on Decision and Control*, 2018, pp. 4238–4245.

[18] A. Olshevsky, I. C. Paschalidis, and S. Pu, "A non-asymptotic analysis of network independence for distributed stochastic gradient descent," *arXiv:1906.02702*, 2019.

[19] P. Di Lorenzo and G. Scutari, "NEXT: In-network nonconvex optimization," *IEEE Trans. Signal Inf. Process. Netw. Process.*, vol. 2, no. 2, pp. 120–136, 2016.

[20] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes," in *54th IEEE Conference on Decision and Control*, 2015, pp. 2055–2060.

[21] G. Qu and N Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Trans. Control of Network Systems*, vol. 5, no. 3, pp. 1245–1260, 2017.

[22] A. Nedić, 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.

[23] R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 315–320, 2018.

[24] M. Zhu and S. Martínez, "Discrete-time dynamic average consensus," *Automatica*, vol. 46(2), pp. 322–329, 2010.

[25] S. A. Alghunaim, K. Yuan, and A. H. Sayed, "A linearly convergent proximal gradient algorithm for decentralized optimization," *arXiv:1905.07996*, 2019.

[26] S. Pu and A. Nedich, "A distributed stochastic gradient tracking method," in *2018 IEEE Conference on Decision and Control*, 2018, pp. 963–968.

[27] 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, 2015.

[28] K. Yuan, B. Ying, X. Zhao, and A. H. Sayed, "Exact diffusion for distributed optimization and learningPart I: Algorithm development," *IEEE Trans. Signal Process.*, vol. 67, no. 3, pp. 708–723, 2018.

[29] R. Xin, U. A. Khan, and S. Kar, "Variance-reduced decentralized stochastic optimization with accelerated convergence," *arXiv preprint arXiv:1912.04230*, 2019.

[30] A. Mokhtari and A. Ribeiro, "DSA: Decentralized double stochastic averaging gradient algorithm," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2165–2199, 2016.

[31] B. Ying, K. Yuan, and A. H. Sayed, "Variance-reduced stochastic learning under random reshuffling," *arXiv:1708.01383*, 2017.

[32] Z. Shen, A. Mokhtari, T. Zhou, P. Zhao, and H. Qian, "Towards more efficient stochastic decentralized learning: Faster convergence and sparse communication," *arXiv:1805.09969*, 2018.

[33] A. Defazio, "A simple practical accelerated method for finite sums," in *Advances in Neural Information Processing Systems*, 2016, pp. 676–684.

[34] H. Hendrikx, F. Bach, and L. Massoulié, "Asynchronous accelerated proximal stochastic gradient for strongly convex distributed finite sums," *arXiv:1901.09865*, 2019.

[35] Q. Lin, Z. Lu, and L. Xiao, "An accelerated randomized proximal coordinate gradient method and its application to regularized empirical risk minimization," *SIAM Journal on Optimization*, vol. 25, no. 4, pp. 2244–2273, 2015.

[36] B. Li, S. Cen, Y. Chen, and Y. Chi, "Communication-efficient distributed optimization in networks with gradient tracking," *arXiv:1909.05844*, 2019.

[37] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[38] K. Yuan, B. Ying, J. Liu, and A. H. Sayed, "Variance-reduced stochastic learning by networked agents under random reshuffling," *IEEE Trans. Signal Process.*, vol. 67, no. 2, pp. 351–366, 2018.

[39] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE, 2003, pp. 482–491.

[40] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 97: 344–353.

[41] C. Xi, R. Xin, and U. A. Khan, "ADD-OPT: Accelerated distributed directed optimization," *IEEE Trans. Autom. Control*, vol. 63, no. 5, pp. 1329–1339, 2017.

[42] R. Xin, C. Xi, and U. A. Khan, "FROST – Fast row-stochastic optimization with uncoordinated step-sizes," *EURASIP Journal on Advances in Signal Processing*, Nov. 2018.

[43] S. Pu, W. Shi, J. Xu, and A. Nedić, "A push-pull gradient method for distributed optimization in networks," in *IEEE Conference on Decision and Control*, Dec. 2018, pp. 3385–3390.

[44] R. Xin, A. K. Sahu, U. A. Khan, and S. Kar, "Distributed stochastic optimization with gradient tracking over strongly-connected networks," in *IEEE Conference on Decision and Control*, 2019.

[45] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," *Mathematical Programming*, pp. 1–48, 2017.

[46] J. Zhang and K. You, "ASYSPA: An exact asynchronous algorithm for convex optimization over digraphs," *IEEE Transactions on Automatic Control*, 2019.

[47] A. Reisizadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, "An exact quantized decentralized gradient descent algorithm," *IEEE Trans. Signal Process.*, vol. 67, no. 19, pp. 4934–4947, 2019.

[48] A. Koloskova, S. U. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," *arXiv preprint arXiv:1902.00340*, 2019.

[49] X. Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[50] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, Apr. 2017, vol. 54, pp. 1273–1282.