Scaling-up Distributed Processing of Data Streams for Machine Learning

Matthew Nokleby, Haroon Raja, and Waheed U. Bajwa, *Senior Member, IEEE*

Abstract

Emerging applications of machine learning in numerous areas—including online social networks, remote sensing, internet-of-things systems, smart grids, and more—involve continuous gathering of and learning from streams of data samples. Real-time incorporation of streaming data into the learned machine learning models is essential for improved inference in these applications. Further, these applications often involve data that are either inherently gathered at geographically distributed entities due to physical reasons—e.g., internet-of-things systems and smart grids—or that are intentionally distributed across multiple computing machines for memory, storage, computational, and/or privacy reasons. Training of machine learning models in this distributed, streaming setting requires solving stochastic optimization problems in a collaborative manner over communication links between the physical entities. When the streaming data rate is high compared to the processing capabilities of individual computing entities and/or the rate of the communications links, this poses a challenging question: how can one best leverage the incoming data for distributed training of machine learning models under constraints on computing capabilities and/or communications rate? A large body of research in distributed online optimization has emerged in recent decades to tackle this and related problems. This paper reviews recently developed methods that focus on large-scale distributed stochastic optimization in the compute- and bandwidth-limited regime, with an emphasis on convergence analysis that explicitly accounts for the mismatch between computation, communication and streaming rates, and that provides sufficient conditions for order-optimum convergence. In particular, it focuses on methods that solve: (i) distributed stochastic convex problems, and (ii) distributed principal component analysis, which is a nonconvex problem with geometric structure that permits global convergence. For such methods, the paper discusses recent advances in terms of distributed algorithmic designs when faced with high-rate streaming data. Further, it reviews theoretical guarantees underlying these methods, which show there exist regimes in which systems can learn from distributed processing of streaming data at order-optimal rates—nearly as fast as if all the data were processed at a single super-powerful machine.

Index Terms

Convex optimization, distributed training, empirical risk minimization, federated learning, machine learning, mini-batching, principal component analysis, stochastic gradient descent, stochastic optimization, streaming data

M. Nokleby (matthew.nokleby@target.com) is with the Target Corporation, Minneapolis, MN. H. Raja (hraja@umich.edu) is with the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, MI. W.U. Bajwa (waheed.bajwa@rutgers.edu) is with the Department of Electrical and Computer Engineering and Department of Statistics at Rutgers University–New Brunswick, NJ.

The work of WUB has been supported in part by the National Science Foundation under awards CCF-1453073, CCF-1907658, and OAC-1940074, by the Army Research Office under award W911NF-17-1-0546, and by the DARPA Lagrange Program under ONR/NIWC contract N660011824020. The work of HR has been supported in part by the National Science Foundation under awards CCF-1845076 and IIS-1838179, and by the Army Research Office under award W911NF-19-1-0027.

I. Introduction

A. Motivation and Background

Over the past decade—and especially the past few years—there has been a rapid increase in research and development of *artificial intelligence* (AI) systems across the public and private sectors. A significant fraction of this increase is attributable to remarkable recent advances in a subfield of AI that is termed *machine learning*. Briefly, a machine learning system uses a number of data samples—referred to as *training data*—in order to *learn* a mathematical *model* of some aspect of the physical world that can then be used for automated decision making; see Fig. 1(a) for an example of this in the context of automated tagging of images of cats and dogs. Training a machine learning model involves mathematical optimization of a *data-driven* function with respect to the model variable. The decision making capabilities of a machine learning system, in particular, tend to be directly tied to one's ability to solve the resulting optimization problem up to a prescribed level of accuracy.

While solution accuracy remains one of the defining aspects of machine learning, the advent of *big data*—in terms of data dimensionality and/or number of training samples—and the adoption of large-scale models with millions of parameters in machine learning methods such as *deep learning* [1] has catapulted the computing time for training (i.e., the *training time*) to another one of the defining parameters of modern systems. It is against this backdrop that stochastic optimization methods such as *stochastic gradient descent* (SGD) and its variants [2]–[5], in which training data are processed one sample or a small batch of samples—referred to as a *mini batch*—per iteration, as opposed to deterministic optimization methods such as gradient descent [6], in which the entire batch of training data is used in each iteration, have become the de facto standard for faster training of models.

Another major shift in machine learning practice concerns the use of *distributed* and *decentralized* computing platforms, as opposed to a single computing unit, for training of models. There are myriad reasons for this paradigm shift, which range from the focus on further decreasing the training times and preserving privacy of data to adoption of machine learning for decision making in inherently decentralized systems. In particular, distributed and decentralized training of machine learning models can be epitomized by the following three prototypical frameworks.

- *Distributed computing framework:* A distributed computing framework, also referred to as a *compute cluster*, brings together a set of computing units such as CPUs and GPUs to accelerate training of large-scale machine learning models from big data in a more cost-effective manner than a single computer with comparable storage capacity, memory, and computing power. Computing units/machines in a compute cluster typically communicate among themselves using either Ethernet or InfiniBand interconnects, with the intra-cluster communication infrastructure often abstracted in the form of a graph in which vertices/nodes correspond to computing units. A typical graph structure that is commonly utilized for distributed training within compute clusters is *star graph*, which corresponds to the so-called *master–worker* architecture; see, e.g., Fig. 1(b). Training data within this setup is split among the worker nodes, which perform bulk of the computations, while the master node coordinates the distributed training of machine learning model among the worker nodes.
- *Federated learning framework:* The term "federated learning," coined in [7], refers to any machine learning setup in which a collection of autonomous entities (e.g., smartphones and hospitals), each maintaining its own

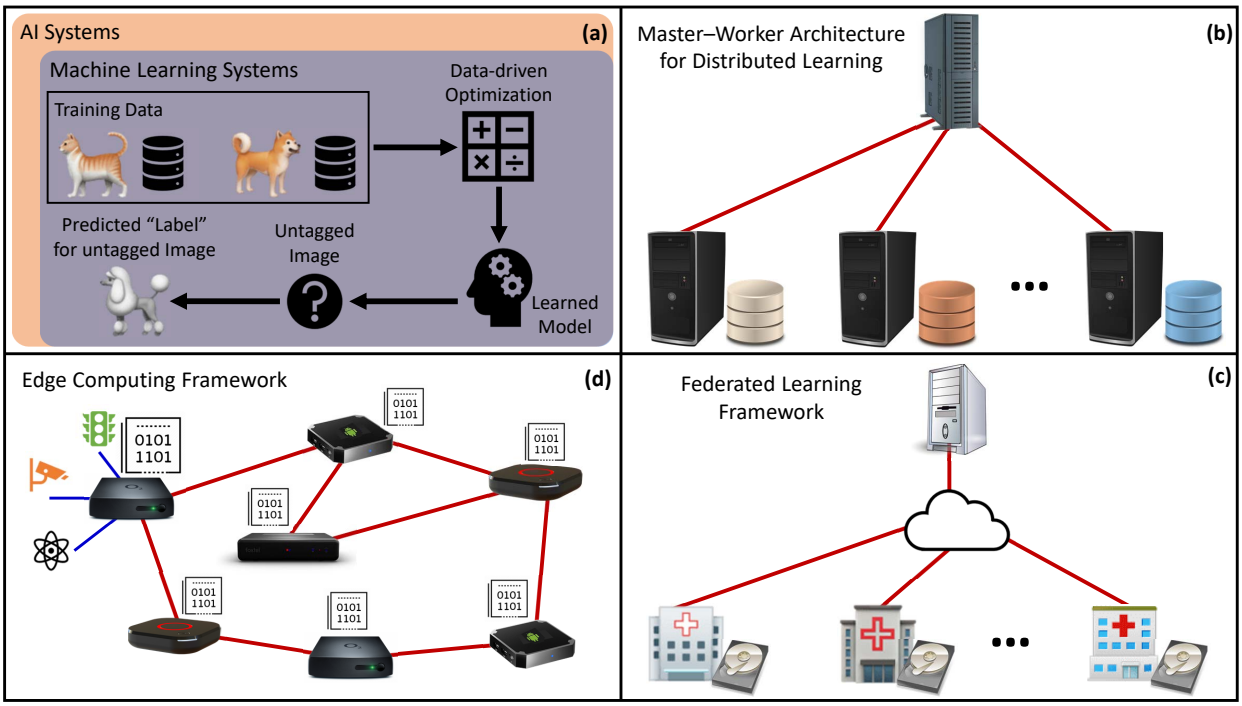


Fig. 1. A schematic quad chart illustrating four different concepts in machine learning. (a) A simplified representation of a machine learning system within the context of an image tagging application. (b) Master–worker architecture that can be used for distributed training of a machine learning model within a compute cluster. (c) A federated machine learning system representing the federation of a group of autonomous hospitals. (d) A representative edge computing framework that can be used for decentralized training of a machine learning model.

private training data, collaborate under the coordination of a central server to learn a "global" machine learning model that best describes the collective non-collocated/distributed training data. A typical federated learning system, in which entities collaborate only through communication with the central server and are prohibited from sharing raw training samples with the server, can also be abstracted as a star graph; see, e.g., Fig. 1(c). However, unlike a master–worker distributed machine learning system—in which the primary objective is reduction of the wall-clock time for training of machine learning models, the first and foremost objective of a federated learning system is to preserve privacy of the data of collaborating entities.

- *Edge computing framework:* The term "edge computing" refers to any decentralized computing system comprising geographically distributed and compact computing devices that collaboratively complete a computational task through local computations and device-to-device communications. Some of the defining characteristics of an edge computing system, which set it apart from a compute cluster, include lack of a coordinating central server, (relatively) slower-speed device-to-device communications (e.g., wireless communications and power-line communications), and abstraction of inter-device communication infrastructure in terms of arbitrary graph topologies (as opposed to star topology). Many emerging edge computing systems, such as the *internet-of-things* (IoT) systems and smart grids, have each computing device connected to a number of data-gathering sensors that generate large volumes of data. Since exchange of these large-scale "local" data among the computing

devices becomes prohibitive due to communication constraints, machine learning in such systems necessitates decentralized collaborative training that involves each device learning (approximately) the same "local" model through inter-device communications that best fits the collective system data; see, e.g., Fig. 1(d).

The purpose of this paper is to provide an overview of some important aspects of distributed/decentralized machine learning that have implications for all three of the aforementioned frameworks. We slightly abuse terminology in the following for ease of exposition and refer to training of machine learning models under any one of these frameworks as distributed machine learning. When one considers distributed training of (large-scale) models from (big) distributed datasets, it raises a number of important questions; these include: (i) What are the fundamental limits on solution accuracy of distributed machine learning? (ii) What kind of optimization frameworks and communication strategies (which exclude exchange of raw data among subcomponents of the system) result in near-optimal distributed learning? (iii) How do the topology of the graph underlying the distributed computing setup and the speed of communication links in the setup impact the learning performance of any optimization framework? A vast body of literature in the last decade has addressed these (and related) questions for distributed machine learning by expanding on foundational works in distributed consensus [8], [9], distributed diffusion [10], distributed optimization [11], [12], and distributed computing [13]. Several of the key findings of such works have also been elucidated through excellent survey articles and overview papers in recent years [14]–[23]. Nonetheless, there remains a need to better understand the interplay between solution accuracy, communication capabilities, and computational resources in distributed systems that carry out training using "streaming" data. Indeed, distributed training using streaming data necessitates utilization of *single-pass* stochastic optimization within the distributed framework, which gives rise to several important operational changes that are not widely known. It is in this regard that this overview paper summarizes some of the key research findings, and their implications, in relation to distributed machine learning from streaming data.

B. Streaming Data and Distributed Machine Learning

Continuous gathering of data is a hallmark of the digital revolution; in countless applications, this translates into streams of data entering into the respective machine learning systems. Within the context of distributed machine learning, the continuous data gathering has the effect of training data associated with each "node" in the distributed system being given in the form of a data stream (cf. Fig. 3 in Section II). Since "(full) batch processing" is practically infeasible in the face of continuous data arrival, distributed training of models from streaming data requires (single-pass) stochastic optimization methods. Accordingly, we provide in this paper an overview of some of the state-of-the-art concerning stochastic optimization-based distributed training from streaming data.

Unlike much of the literature on centralized machine learning from streaming data, (relative) streaming rate of data—defined as the (average) number of new data samples arriving per second—fundamentally shapes the discussion of streaming-based distributed machine learning. In this regard, our objective is to elucidate the performance challenges and fundamental limits when the streaming rate of data is *fast* compared to the processing speed of computing units and/or the communications speed of inter-node links in the system. In particular, this involves addressing of the following question: *what happens to the solution accuracy of distributed machine learning when it*

is impossible to have high-performance computing machines for computing nodes and/or (multi-)gigabit connections for inter-node communication links? Note that this question cannot be addressed by simply "slowing down" the data stream(s) through regular discarding of incoming samples. Within a distributed computing framework, for instance, letting some of the incoming samples pass without updating the model would be antithetical to its overarching objective of accelerated training. Similarly, downsampling of time-series data streams in an edge computing system would cause the system to lose out on critical high-frequency modes of data. In short, processing all data samples arriving into the distributed system and incorporating them into the learned model is both paramount and non-trivial.

There are many ways to frame and analyze the problem of distributed machine learning from fast streaming data, leading to far more relevant works than we can discuss in this overview paper. Instead, we provide a very brief discussion of the different framings, and motivate our prioritization of the following system choices under the general umbrella of distributed machine learning: **decentralized-parameter** systems, **synchronous-communications** distributed computing, and **statistical risk minimization** for training of machine learning models. We dive into the relevant distinctions for these system choices in the following.

C. General Framing of the Overview

The area of distributed machine learning is far too rich and broad to be covered in a paper. Instead, we cover only some aspects of the area that are the most relevant to the topic of distributed machine learning from streaming data. To put the rest of our discussion in context, we give a very coarse description of these aspects in the following, drawing out some of the crucial distinctions and pointing out which aspects remain uncovered in the paper.

System models for distributed learning. We abstract away the dependence on any particular computing architecture by modeling the architecture as an interconnected network of (computing) nodes having a certain topology (e.g., star topology for the master–worker architecture). Accordingly, our discussion is applicable to any of the computing frameworks discussed in Section I-A that adhere to the data and system assumptions described later in Section II. In the interest of generality, we also move away from the so-called *parameter-server* system model that is used in some distributed environments [24], [25]. In the simplest version of this model, a single node—termed parameter server—maintains and updates parameters of the machine learning model, whereas the remaining nodes in the network compute gradients of their local data that are then transmitted to the parameter server and used to make updates to the shared set of parameters. We instead center our discussion around the *decentralized-parameter* system model, where each node maintains and updates its own copy of the parameters. This system model is more general, since any result that holds for a decentralized-parameter network also holds for a parameter-server network, it prevents a single point of failure in the system, and it allows us to present a unified discussion that transcends multiple system models.

Models for message passing and communications. Algorithmic-level synchronization (or lack thereof) among different computing nodes is one of the most important design choices in distributed implementations. On one hand, *synchronous* implementations (which often make use of "blocking" message passing protocols for synchronization [26], [27]) can slow down training times due to either message passing (i.e., communications) delays or "straggler" nodes taking longer than the rest of the network to complete their subtasks. On the other hand,

asynchronous implementations have the potential to drastically impact the solution accuracy. Such tradeoffs between synchronous and asynchronous implementations, as well as approaches that hybridize the two, have been investigated in recent years [28]–[33]. In this paper, we focus exclusively on synchronous implementations for the sake of concreteness. In addition, we abstract lower-level communications within the synchronous system as happening in discrete, pre-defined epochs (time intervals, slots, etc.). While such an abstraction models only a restrictive set of communications protocols, it greatly simplifies the exposition without sacrificing too much of the generality.

Optimization framework for distributed machine learning. Machine learning problems involve the optimization of a "loss" function with respect to the machine learning model. And this optimization side of machine learning can be framed in two major interrelated ways. The first (and perhaps most well-known) framing is referred to as *empirical risk minimization* (ERM). The objective in this case is to minimize the *empirical* risk $\hat{f}(\mathbf{w})$, defined as the empirical average of the so-called *(regularized) loss function* $\ell(\mathbf{w}, \cdot)$ evaluated on the training samples, with respect to the model variable $\mathbf{w}$. Under mild assumptions on the loss function, data distribution, and training data, the ERM solution $\mathbf{w}^*_{\text{ERM}} \in \arg\min_{\mathbf{w}} \hat{f}(\mathbf{w})$ is known to converge (with high probability) to the minimizer $\mathbf{w}^*$ of the "true" risk $f(\mathbf{w})$, i.e., the *expected* loss $f(\mathbf{w}) := \mathbb{E}[\ell(\mathbf{w}, \cdot)]$ [34], with study of the rates of this convergence being a long-standing and active research area [35]. Distributed learning literature within the ERM framework typically supposes a fixed and finite number T of training samples distributed across computing nodes, and primarily focuses on understanding convergence of the output $\widehat{\mathbf{w}}^*_{\text{ERM}}$ of different *distributed optimization* schemes to the ERM solution $\mathbf{w}^*_{\text{ERM}}$ [19]. The accuracy of the final solution, termed *excess risk* and defined as $f(\widehat{\mathbf{w}}^*_{\text{ERM}}) - f(\mathbf{w}^*)$, is then provided either implicitly or explicitly in the works as the sum of two gaps: (i) gap between the risk of the distributed optimization solution and that of the ERM solution, i.e., $f(\widehat{\mathbf{w}}^*_{\text{ERM}}) - f(\mathbf{w}^*_{\text{ERM}})$, and (ii) gap between the risk of the ERM solution and that of the optimal solution, termed *Bayes' risk*, i.e., $f(\mathbf{w}^*_{\text{ERM}}) - f(\mathbf{w}^*)$. In contrast to the ERM framework, the second optimization-based framing of machine learning—termed *statistical risk minimization* (SRM)—facilitates a direct bound on the excess risk; see, e.g., Fig. 2. This is because the objective in SRM framework is to minimize expected loss (risk) over the true data distribution, as opposed to empirical loss over the training data in ERM framework. The SRM framework falls squarely within the confines of stochastic optimization, with a large body of existing work—covering both centralized and distributed machine learning—that characterizes the excess risk of the resulting solution $\mathbf{w}_{\text{SRM},T}$ under the assumption that either the number of training samples T is sufficiently large or it grows asymptotically. Since we are concerned with streaming data, in which a virtually unbounded number of samples may arrive at the system, we focus on the SRM-based framework and single-pass stochastic optimization for distributed machine learning. We discuss further the distinction between the convergence results derived under the frameworks of ERM and SRM in the sequel.

Structure of the optimization objective function. The vast majority of works at the intersection of (stochastic) optimization and (distributed) learning suppose that the loss function is *convex* with respect to the model parameters. But some of the most exciting recent results in machine learning have come about in the context of deep learning, where the objective function tends to be highly nonconvex and most practical methods do not even concern themselves with *global optimality* of the solution [36], [37]. Nevertheless, for the purpose of being able to carry out analysis, we focus in this overview on either convex problems or *structured* nonconvex problems, such as *principal*

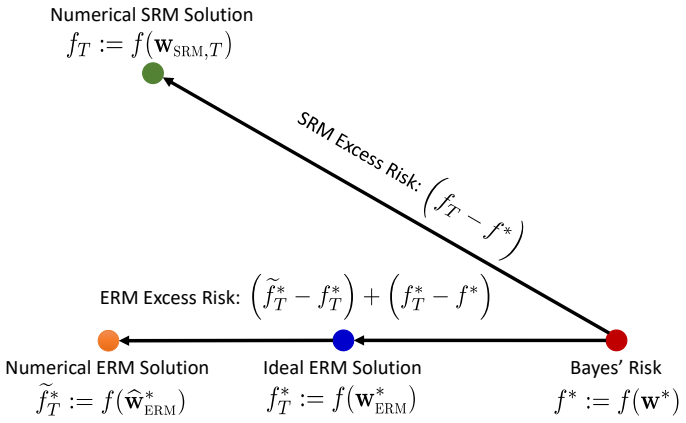


Fig. 2. A geometrical view of "excess risk," defined as the gap between the expected loss $f(\mathbf{w})$ (i.e., risk) of the solution $\mathbf{w}$ of a practical machine learning algorithm and that of an ideal solution $\mathbf{w}^*$ (i.e., Bayes' risk $f^* := f(\mathbf{w}^*)$), under the ERM and SRM optimization frameworks after receiving a total of T training data samples. Under the ERM framework, the excess risk of the solution is bounded as the sum of bounds on two terms, namely, the gap between the risk of an ideal ERM solution and Bayes' risk, $(f_T^* - f^*)$, and the gap in risk due to the error associated with numerical optimization, $(\widetilde{f}_T^* - f_T^*)$. In contrast, excess risk under the SRM framework is directly captured in terms of the gap between the risk of the numerical SRM solution $\mathbf{w}_{\mathrm{SRM},T}$ and Bayes' risk, i.e., $(f_T - f^*)$.

component analysis (PCA), where the structure can be exploited by local search methods to find a global solution.

D. An Outline of the Overview Paper

We now provide an outline of the remainder of this paper. Section II gives a formal description of the learning and system models considered in this paper, including the loss function, the streaming data model, the communications model, and the way compute nodes exchange messages with each other during distributed learning. In Section III, we discuss relevant results in (distributed) machine learning that prefigure the state-of-the-art being reviewed in the paper. Section IV and Section V of the paper are devoted to coverage of the state-of-the-art in terms of distributed machine learning from fast streaming data. The main distinction between the two sections is the nature of the communications infrastructure underlying the distributed computing framework. Section IV focuses on the case of (relatively) high-speed communications infrastructure that enables completion of message-passing primitives such as `AllReduce` [27] in a "reasonable" amount of time, whereas Section V discusses distributed machine learning from streaming data in systems with (relatively) lower-speed communications infrastructure. In both cases, we discuss scenarios and distributed algorithms that can lead to near-optimal excess risk for the final solution as a function of the number of data samples arriving at the system; in addition, we present results of numerical experiments to corroborate some of the stated results. One of the key insights delivered by these two sections is that a judicious use of (implicit or explicit) mini-batching of data samples in distributed systems is fundamental in dealing with fast streaming data in compute- and/or communications-limited scenarios. To this end, we provide theoretical results for the optimum choice of the size of (network-wide and local) mini-batches as well as conditions on when mini-batching is sufficient to achieve near-optimal convergence. We conclude the paper in Section VI with a brief recap

of the implications of presented results for the practitioners as well as a discussion of possible next steps for researchers working on distributed machine learning.

E. Notational Convention

We use regular-faced (e.g., a and B), bold-faced lower-case (e.g., $\mathbf{a}$), and bold-faced upper-case (e.g., $\mathbf{A}$) letters for scalars, vectors, and matrices, respectively. We use calligraphic letters (e.g., $\mathcal{A}$) to represent sets, while $[\![N]\!] := \{1, \ldots, N\}$ denotes the set of first N natural numbers, and $\mathbb{R}_{\geq 0}$ and $\mathbb{Z}_+$ denote the sets of non-negative real numbers and positive integers, respectively. Given a vector $\mathbf{a}$ and a matrix $\mathbf{A}$, $\|\mathbf{a}\|_2 := \sqrt{\sum_i |a_i|^2}$, $\|\mathbf{A}\|_2 := \arg\max_{\mathbf{v}:\mathbf{v}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{v}\|_2}{\|\mathbf{v}\|_2}$, and $\|\mathbf{A}\|_F := \sqrt{\sum_{i,j} |a_{ij}|^2}$ denote the ℓ_2-norm of $\mathbf{a}$, the spectral norm of $\mathbf{A}$, and the Frobenius norm of $\mathbf{A}$, respectively. Given a symmetric matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$, $\lambda_i(\mathbf{A})$ denotes its i-th largest-by-magnitude eigenvalue, i.e., $|\lambda_1(\mathbf{A})| \geq \cdots \geq |\lambda_d(\mathbf{A})| \geq 0$. Given a function $f : \mathbb{R}^d \times \mathcal{Z} \to \mathbb{R}$ that is partially differentiable in the first argument, ∇f denotes the gradient of $f(\cdot, \cdot)$ with respect to its first argument. Given functions $f(x)$ and $g(x)$, we use Landau's *Big-O* notation (e.g., $f(x) = O(g(x))$ or $f(x) = o(g(x))$) to describe the scaling relationship between them. Finally, $\mathbb{E}\{\cdot\}$ denotes the expectation operator, where the underlying probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is either implicit from the context or is explicitly noted.

II. Problem Formulation

In this section, we discuss the problem of distributed processing of fast streaming data for machine learning in three parts. First, we describe the general statistical optimization problem underlying machine learning. Second, we describe the system model that formalizes distributed processing of streaming data . Finally, we formalize the notion of *fast* streaming data in terms of, among other things, data streaming rate, processing rate of compute nodes, and communications rate of inter-node links within the distributed environment.

A. Statistical Optimization for Machine Learning

Most machine learning problems can be posed as data-driven optimization problems, with the objective termed a *loss function* that quantifies the error (classification, regression or clustering error, mismatch between the learned and true data distributions, etc.) in a candidate solution. We denote this loss function by $\ell : \mathcal{W} \times \mathcal{Z} \to \mathbb{R}_{\geq 0}$, where $\mathcal{W}$ denotes the space of candidate machine learning models and $\mathcal{Z}$ denotes the space of data samples. Given a model $\mathbf{w} \in \mathcal{W}$, $\ell(\mathbf{w}, \mathbf{z})$ measures the modeling loss associated with $\mathbf{w}$ in relation to the data sample $\mathbf{z} \in \mathcal{Z}$.[1]

Several examples of loss functions and their respective model space(s) for *supervised learning* (e.g., regression and classification) and *unsupervised learning* (e.g., feature learning and clustering) problems are listed below.

Loss functions and models for supervised machine learning. Data samples in supervised learning can be expressed as tuples $\mathbf{z} := (\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R} =: \mathcal{Z}$, with $\mathbf{x}$ referred to as data and y referred to as its *label*. In particular, focusing on the linear classification problem with label $y \in \{-1, 1\}$, augmented data $\widetilde{\mathbf{x}} := [\mathbf{x}^{\mathrm{T}}\ 1]^{\mathrm{T}} \in \mathbb{R}^{d+1}$, and model

[1]The model space $\mathcal{W}$ in most formulations is taken to be one that is completely described by a set of parameters. For example, if $\mathcal{W}$ denotes the space of all polynomials of degree $(d-1)$, a model $\mathbf{w} \in \mathcal{W}$ is uniquely characterized by the d coefficients of the respective polynomial. In this paper, we slightly abuse the notation and use $\mathbf{w}$ to denote both the model and, when the model is parameterizable, its respective parameters.

space $\mathcal{W} := \mathbb{R}^{d+1}$, the model $\mathbf{w}$ describes an affine hyperplane in $\mathbb{R}^d$ and two common choices of loss functions are: (i) *Hinge loss:* $\ell(\mathbf{w}, \mathbf{z}) := \max\left(0, 1 - y \cdot \mathbf{w}^{\mathrm{T}}\widetilde{\mathbf{x}}\right)$, and (ii) *Logistic loss:* $\ell(\mathbf{w}, \mathbf{z}) := \ln\left(1 + \exp\left(-y \cdot \mathbf{w}^{\mathrm{T}}\widetilde{\mathbf{x}}\right)\right)$.

Loss functions and models for unsupervised machine learning. Data samples in unsupervised machine learning do not have labels, with the *unlabeled* data sample $\mathbf{z} \in \mathbb{R}^d =: \mathcal{Z}$ in this case. We now describe the loss functions and models/model parameters associated with two popular unsupervised machine learning problems.

- *Principal component analysis (PCA):* The k-PCA problem is a feature learning problem in which the model space is $\mathcal{W} := \left\{\mathbf{A} \in \mathbb{R}^{d\times k} : \mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{I}\right\}$, the matrix-valued model $\mathbf{W} \in \mathcal{W}$ describes a k-dimensional subspace of $\mathbb{R}^d$, and the loss function is $\ell(\mathbf{W}, \mathbf{z}) := \|\mathbf{z} - \mathbf{W}\mathbf{W}^{\mathrm{T}}\mathbf{z}\|_2^2$.
- *Center-based clustering:* The k-means clustering problem has the model space $\mathcal{W} := \underbrace{\mathbb{R}^d \times \cdots \times \mathbb{R}^d}_{k \text{ times}}$, the model $(\mathbf{w}_1, \ldots, \mathbf{w}_k) \in \mathcal{W}$ is a k-tuple of d-dimensional *cluster centroids*, and a common choice of the loss function is $\ell\left((\mathbf{w}_1, \ldots, \mathbf{w}_k), \mathbf{z}\right) := \min_{1\le i\le k} \|\mathbf{z} - \mathbf{w}_i\|_2^2$.

In this paper, our discussion of machine learning revolves around the statistical learning viewpoint [34]. To this end, we suppose each data sample $\mathbf{z}$ is drawn from some unknown probability distribution $\mathcal{D}$ that is supported on $\mathcal{Z}$. The overarching goal in (statistical) machine learning then is to obtain a model $\mathbf{w} \in \mathcal{W}$ that has the smallest loss *averaged over all* $\mathbf{z} \in \mathcal{Z}$. Specifically, let

$$f(\mathbf{w}) := \mathbb{E}_{\mathbf{z}\sim\mathcal{D}}\left\{\ell(\mathbf{w}, \mathbf{z})\right\} \tag{1}$$

denote the expected loss, also referred to as *(statistical) risk*, associated with model $\mathbf{w}$ for the entire data space $\mathcal{Z}$. Then, the objective of machine learning from the statistical learning perspective is to approach the *Bayes optimal* solution $\mathbf{w}^*$ that minimizes the statistical risk, i.e.,

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}\in\mathcal{W}} f(\mathbf{w}). \tag{2}$$

The risk incurred by $\mathbf{w}^*$ (i.e., $f(\mathbf{w}^*)$) is termed *Bayes' risk*. The main challenge in machine learning is that the distribution $\mathcal{D}$ is unknown and therefore (2) cannot be directly solved. Instead, one uses training data samples $\{\mathbf{z}_{t'}\}_{t'\in\mathbb{Z}_+}$ that are independently drawn from $\mathcal{D}$ to obtain a model $\widehat{\mathbf{w}}$ whose risk comes close to Bayes' risk as a function of the number of training samples. In particular, the performance of a machine learning algorithm is measured in terms of either the *excess risk* of its solution, defined as $f(\widehat{\mathbf{w}}) - f(\mathbf{w}^*)$, or the *parameter estimation error* calculated in terms of some distance between the solution $\widehat{\mathbf{w}}$ and the set of minimizers $\arg\min_{\mathbf{w}\in\mathcal{W}} f(\mathbf{w})$.

Since optimization is central to machine learning, the geometrical structure and properties of the loss function determine whether and how easily a method finds a solution $\widehat{\mathbf{w}}$ that has (nearly) minimal excess risk / estimation error. We describe this structure and properties of $\ell(\mathbf{w}, \mathbf{z})$ in terms of its *gradients*, *convexity*, and *smoothness*.

Definition 1 (Existence of Gradients)**.** A loss function $\ell(\mathbf{w}, \mathbf{z})$ is said to have its gradients exist everywhere if $\nabla_{\mathbf{w}}\ell(\mathbf{w}, \mathbf{z})$ exists for all $(\mathbf{w}, \mathbf{z}) \in \mathcal{W} \times \mathcal{Z}$.

Definition 2 (Convexity and Strong Convexity)**.** A loss function $\ell(\mathbf{w}, \mathbf{z})$ is *convex* in $\mathbf{w}$ if for all $\mathbf{w}_1, \mathbf{w}_2 \in \mathcal{W}$, all $\mathbf{z} \in \mathcal{Z}$, and all $\alpha \in [0, 1]$, we have

$$\ell(\alpha\mathbf{w}_1 + (1-\alpha)\mathbf{w}_2, \mathbf{z}) \le \alpha\ell(\mathbf{w}_1, \mathbf{z}) + (1-\alpha)\ell(\mathbf{w}_2, \mathbf{z}).$$

In words, the function $\ell(\cdot, \mathbf{z})$ for all $\mathbf{z} \in \mathcal{Z}$ must lie below any chord for the loss function to be convex in $\mathbf{w}$. Further, a loss function whose gradients exist everywhere is said to be *strongly convex* with modulus $m > 0$ if for all $\mathbf{w}_1, \mathbf{w}_2 \in \mathcal{W}$ and all $\mathbf{z} \in \mathcal{Z}$, we have

$$(\nabla_{\mathbf{w}} \ell(\mathbf{w}_1, \mathbf{z}) - \nabla_{\mathbf{w}} \ell(\mathbf{w}_2, \mathbf{z}))^{\mathrm{T}} (\mathbf{w}_1 - \mathbf{w}_2) \geq m \|\mathbf{w}_1 - \mathbf{w}_2\|_2^2.$$

Definition 3 (Smoothness)**.** We say that a loss function whose gradients exist everywhere is *smooth* if its gradients are Lipschitz continuous with some constant $L > 0$, i.e., for all $\mathbf{w}_1, \mathbf{w}_2 \in \mathcal{W}$ and all $\mathbf{z} \in \mathcal{Z}$, we have

$$\|\nabla_{\mathbf{w}} \ell(\mathbf{w}_1, \mathbf{z}) - \nabla_{\mathbf{w}} \ell(\mathbf{w}_2, \mathbf{z})\|_2 \leq L \|\mathbf{w}_1 - \mathbf{w}_2\|_2.$$

Going forward, we drop the subscript $\mathbf{w}$ in $\nabla_{\mathbf{w}} \ell(\mathbf{w}, \mathbf{z})$ for notational compactness. Note that in the case of a smooth, convex (loss) function, gradient-based local search methods are guaranteed to converge to a global minimizer of the function. In addition, the global minimizer is unique for *strongly* convex functions and convergence of gradient-based methods to the minimizer of these functions is provably fast.

Our discussion revolves around both convex and (certain structured) nonconvex loss functions. Some of it in relation to convex losses requires an assumption on the variance of the gradients with respect to the data distribution.

Definition 4 (Gradient Noise)**.** We say the gradients of $\ell(\cdot, \mathbf{z})$ have *bounded variance* if for every $\mathbf{w} \in \mathcal{W}$, we have

$$\mathbb{E}_{\mathbf{z} \sim \mathcal{D}} \left\{ \|\nabla \ell(\mathbf{w}, \mathbf{z}) - \nabla f(\mathbf{w})\|_2^2 \right\} \leq \sigma^2.$$

In the following, we term σ^2 as the *gradient noise variance*. In addition, we use the notion of *single-sample covariance noise variance* in lieu of gradient noise variance in relation to our discussion of the nonconvex loss function associated with the 1-PCA problem.

Definition 5 (Sample-covariance Noise)**.** We say the single-sample covariance matrix $\mathbf{z}\mathbf{z}^{\mathrm{T}}$ associated with data sample $\mathbf{z}$ drawn from distribution $\mathcal{D}$ has *bounded variance* if we have

$$\mathbb{E}_{\mathbf{z} \sim \mathcal{D}} \left\{ \left\| \mathbf{z}\mathbf{z}^{\mathrm{T}} - \mathbb{E}_{\mathbf{z} \sim \mathcal{D}} \left\{ \mathbf{z}\mathbf{z}^{\mathrm{T}} \right\} \right\|_F^2 \right\} \leq \sigma^2.$$

The gradient (resp., sample covariance) noise variance controls the error associated with evaluating the gradient (resp., sample covariance) at individual sample points $\mathbf{z}$ instead of evaluating it at the statistical mean of the unknown distribution $\mathcal{D}$. Smaller gradient (resp., sample covariance) noise variance results in faster convergence, and the main message of this paper is that leveraging distributed streams to average out gradient (resp., sample covariance) noise is often an optimum way to speed up convergence in compute- and/or communications-limited regimes.

The last definition we need is that of a *bounded model space*, which plays a role in the analysis of optimization methods for convex loss functions.

Definition 6 (Bounded model space)**.** Let $D_{\mathcal{W}} := \sqrt{\max_{\mathbf{u}, \mathbf{v} \in \mathcal{W}} \|\mathbf{u} - \mathbf{v}\|_2^2 / 2}$ denote the *expanse* of the model space $\mathcal{W}$. We say that an optimization problem has bounded model space if $D_{\mathcal{W}} < \infty$.

Since our focus is training from fast streaming data that necessitates distributed processing, we next formalize the distributed processing / communications framework underlying the algorithms being discussed in the paper.

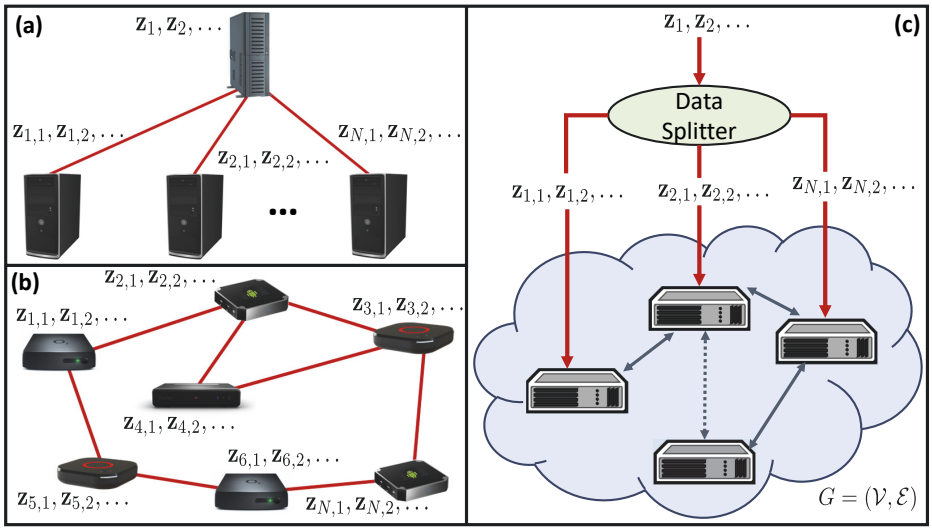


Fig. 3. Distributed training of machine learning models from streaming data can arise in several contexts, including (a) the master–worker computing framework and (b) the edge computing and federated learning frameworks. In this paper, we study a unified abstraction (c) of such frameworks, in which a data stream is split into N parallel streams, one for each compute node in a network $G = (\mathcal{V}, \mathcal{E})$ of N nodes.

B. Distributed Training of Machine Learning Models from Streaming Data

In addition to optimality, in the face of large volumes and high dimensionality of data in modern applications, the solution needs to be efficient in terms of resource utilization as well (e.g., computational, communication, storage, energy, etc.). In Section I, we discussed three mainstream distributed frameworks for resource-efficient machine learning, where each of the frameworks is primarily designed to adhere to specific practical constraints posed due to characteristics of the training data. One such characteristic is the physical locality of data, which results in following two common scenarios involving streaming data: (i) for the *master–worker* learning framework, the data stream arrives at a single master node and, in order to ease the computational load and accelerate training time, the data stream is then divided among a total of N worker nodes (Fig. 1(b) and Fig. 3(a)), or (ii) for the *federated learning* and *edge computing* frameworks, there is a collection of N geographically distributed nodes—each of which receives its own independent stream of data—and the goal is to learn a machine learning model using information from all these nodes (Fig. 1(c), Fig. 1(d), and Fig. 3(b)). Despite the apparent physical differences between these two scenarios, we can study them under a unified abstraction that assumes the data are arriving at a *hypothetical* "data splitter" that then evenly distributes the data across an interconnected network of N nodes for distributed processing (Fig. 3(c)).

Mathematically, let us discretize the data arrival time as $t' = 1, 2, \ldots$, and let $\mathbf{z}_{t'}$ be a stream of independent and identically distributed (i.i.d.) data samples arriving at the splitter at a fixed rate of R_s samples per second. The splitter then evenly distributes the data stream across a network of N nodes, which we represent by an undirected connected graph $G := (\mathcal{V}, \mathcal{E})$; here, $\mathcal{V} := [\![N]\!]$ denotes the set of all nodes in the network and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges corresponding to the communication links between these nodes, i.e., $(n, k) \in \mathcal{E}$ means there is a

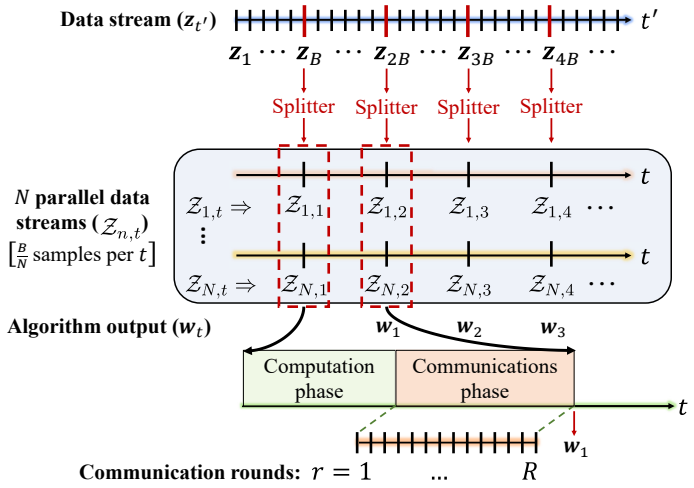


Fig. 4. Data arrival, splitting, processing, and communications timelines within the distributed mini-batch framework of the paper. A stream of data, $\mathbf{z}_{t'}$, arriving at a splitter at the rate of R_s samples per second, is evenly split every B samples across N nodes in the network. This results in *mini-batched* data streams $\mathcal{Z}_{n,t} := \{\mathbf{z}_{n,b,t}\}_{b=1,t\in\mathbb{Z}_+}^{B/N}$ within the network, with $\mathcal{Z}_{n,t}$ denoting the t-th mini-batch of B/N samples at node n. The system processes these distributed mini-batches by engaging in local computations followed by R rounds of inter-node communications and produces an output $\mathbf{w}_t$ before the arrival of the next set of mini-batches.

communication link between nodes n and k. We also define $t \in \mathbb{Z}_+$ to be the index that denotes the total number of data-splitting operations that have been performed within the system. Without loss of generality, we take each data-splitting round to be the time in which nodes carry out a single iteration of a distributed algorithm; i.e., after t data-splitting rounds, the nodes have carried out t iterations of the distributed algorithm under study.

We next set the notation for the distributed data streams within our data-splitting abstraction to facilitate the prevalent practice of training using mini-batches of data samples. To this end, we assume without loss of generality that a total of $B \in \{N, 2N, 3N, \dots\}$ samples arrive at the network during each data-splitting round. That is, a *system-wide* mini-batch of size B is processed by the network during each algorithmic iteration (see, e.g., Fig. 4). Hence, each splitting operation results in a mini-batch of size $\frac{B}{N} \in \mathbb{Z}_+$ arriving at each node. The data splitting across N nodes in the system therefore gives rise to N i.i.d. streams of *mini-batched* data, where we denote the $\frac{B}{N}$ i.i.d. data samples within the t-th mini-batch at node n as $\left\{\mathbf{z}_{n,b,t} \overset{\text{i.i.d.}}{\sim} \mathcal{D}\right\}_{b=1,t\in\mathbb{Z}_+}^{B/N}$, with the mapping of these samples to the ones in the original data stream $\mathbf{z}_{t'}$ given in terms of the relationship $t' = b+(n-1)B/N+(t-1)B$.

Given this distributed, streaming data model, our goal is study of machine learning algorithms that can efficiently process and incorporate the B newest-arriving network-wide samples into a running approximation of the Bayes optimal solution (cf. (2)) *before* the arrival of the next mini-batch of data. In order to highlight the challenges involved in the designs of such algorithms, we can divide the task of processing of a mini-batch of B samples within the network into two phases (cf. Fig. 4): (i) the *computation phase*, in which each node performs computations over its *local* mini-batch of B/N data samples, and (ii) the subsequent *communications phase*, in which nodes share the outcomes of their local computations with each other for eventual incorporation into the (network-wide, decentralized) machine learning models $\{\mathbf{w}_{n,t}\}_{n\in\mathcal{V}}$.

Consider now the *compute-limited* regime within our framework, in which the distributed system comprising N compute nodes is incapable of finishing computations on B samples between two consecutive data-splitting instances because of the fast data streaming rate. (Indeed, the time between two data-splitting instances decreases as R_s increases.) One could push the system out of this compute-limited regime by adding more compute nodes to the system. Keeping the system-wide mini-batch size B fixed (and large), this will result in smaller local mini-batch size B/N. Alternatively, keeping the local mini-batch size fixed, this will result in larger time between two data-splitting instances. And in either case, the system no longer remains compute limited. However, as one adds more and more compute nodes into the distributed system, it could be pushed into the *communications-limited* regime, in which the size/topology of the network prevents the nodes from completing full exchange of their local computations between two consecutive data-splitting instances. This communications-limited regime—which becomes especially pronounced in systems with slower communications links—can only be mitigated through larger data-splitting intervals, which in turn necessitates a larger B for any fixed data streaming rate R_s. But this can again push the system into the compute-limited regime. Therefore, any machine learning algorithm intending to process fast streaming data in an optimal fashion must strike a balance between the compute- and the communications-limited regimes through judicious choices of system parameters such as B and N. We now formalize some of this discussion in the following, which should lead to a better understanding of the interplay between the data streaming rate, the computational capabilities of compute nodes, the communications capabilities of the network, the system-wide mini-batch size B, and the number of compute nodes N in distributed systems.

C. Interplay Between System Parameters in Distributed, Streaming Machine Learning

We have already defined R_s as the number of data samples $\mathbf{z}_{t'}$ arriving per second at the splitter. We also assume the N compute nodes in the system to be homogenous in nature and use R_p to denote the processing/compute rate of each of these nodes, defined as the number of data samples per second that can be locally processed per node during the computation phase. Distributed algorithms also involve the use of message passing routines for inter-node communications. We use R_c to denote the rate of messages shared among nodes using such routines, defined as the number of messages (synchronously) communicated between nodes per second during the communications phase. This parameter R_c also subsumes within itself any overhead associated with implementation of the message passing routine such as time spent on additional computations or communications necessitated by the implementation.

Distributed machine learning algorithms typically involve multiple message passing rounds within the communications phase (cf. Fig. 4), which we denote by $R \in \mathbb{Z}_+$. This parameter R, which we assume remains fixed for the duration of the training, can be constrained in terms of the system parameters B, N, R_s, R_p, and R_c as follows:

$$0 < R \leq \left\lfloor BR_c \left(\frac{1}{R_s} - \frac{1}{NR_p} \right) \right\rfloor. \tag{3}$$

Our focus in this paper is on algorithms that make use of either "exact" or "inexact" *distributed averaging* procedures within the communications phase for information sharing. Specifically, let $\{\mathbf{v}_n \in \mathbb{R}^d\}_{n \in \mathcal{V}}$ be a set of vectors that is distributed across the N nodes in the network at the start of any communications phase and define $\widehat{\mathbf{v}}_n$ to be an

estimate of the average $\bar{\mathbf{v}} := \frac{1}{N}\sum_{n\in\mathcal{V}} \mathbf{v}_n$ of these vectors at node n. We then have the following communications-related characterizations of the algorithms being studied in the paper.

1) **Exact averaging algorithms.** After R message passing rounds within the communications phase, these algorithms can exactly estimate the average at each node, i.e., $\forall n \in \mathcal{V}, \|\widehat{\mathbf{v}}_n - \bar{\mathbf{v}}\|_2 = 0$.
2) **Inexact averaging algorithms.** After R message passing rounds within the communications phase, these algorithms can only guarantee ϵ-accurate estimates, i.e., $\forall n \in \mathcal{V}, \|\widehat{\mathbf{v}}_n - \bar{\mathbf{v}}\|_2 \leq \epsilon$ for some parameter $\epsilon > 0$ that typically increases as R decreases and/or N increases.

Exact averaging algorithms often find applications in settings like high-performance computing clusters and enterprise cloud computing systems, where communications is typically fast and reliable. In contrast, inexact averaging algorithms tend to be more prevalent in settings like edge computing systems, multiagent systems, and IoT systems, where the network connectivity can be sparser and the communications tend to be slower and unreliable.

We have now described all the system parameters needed to formalize the notion of *effective (mini-batch) processing rate*, R_e, of the distributed system, which is defined as the number of mini-batches comprising B samples that can be processed by the system per second. (In the non-distributed setting, corresponding to $N = 1$, it is straightforward to see that $R_e = R_p/B$.) Under the assumption of a synchronous system in which computation and communications phases are carried out one after the other, the parameter R_e can be defined as follows:

$$R_e := \frac{1}{\text{time spent in computation} + \text{time spent in communication}} = \frac{1}{\frac{B}{NR_p} + \frac{R}{R_c}} = \left(\frac{B}{NR_p} + \frac{R}{R_c}\right)^{-1}. \quad (4)$$

This expression formally highlights the tradeoff between the compute-limited and the communications-limited regimes. In the case of fixed R_p and R_c, for instance, increasing the effective processing rate requires an increase in N. Doing so, however, necessitates an increase in R that—beyond a certain point—can only be accomplished through an increase in B (cf. (3)), which in turn also increases the first term in (4).

The overarching theme of this paper is discussion of algorithmic strategies that can be used to tackle the challenge of *near-optimal* training of machine learning models from fast streaming data, where "fast" is defined in the sense that $\frac{R_s}{R_e} \gg B$. This discussion involves allowable selections of system parameters such as the network-wide mini-batch size B, the number of nodes N, and the number of communications rounds R that facilitate taming of the fast incoming data stream without compromising the fidelity of the final solution. In particular, the recommended strategies end up pushing the ratio $\frac{R_s}{R_e}$ to satisfy either $\frac{R_s}{R_e} \leq B$ or $\frac{R_s}{R_e} = (B + \mu)$ for an appropriate parameter $\mu \in \mathbb{Z}_+$, where the latter scenario involves discarding of μ samples per splitting instance t at the data splitter.

In order to prime the reader for subsequent discussion, we also provide a simple example in Fig. 5 that illustrates the impact of the choice of (network-wide) mini-batch size B on system performance. We suppose a network of $N = 10$ compute nodes, and focus on the exact averaging paradigm described above. We assume a data streaming rate of $R_s = 10^6$ samples per second, whereas the data processing rate per node is taken to be $R_p = 1.25 \times 10^5$ samples per second. We plot the ratio of the streaming rate and the effective (mini-batch) processing rate R_e as defined in (4), for communications rates $R_c = 10^3$ and $R_c = 10^4$, as a function of the mini-batch size B. As noted earlier, the number of samples effectively processed by the network keeps pace with the number of samples

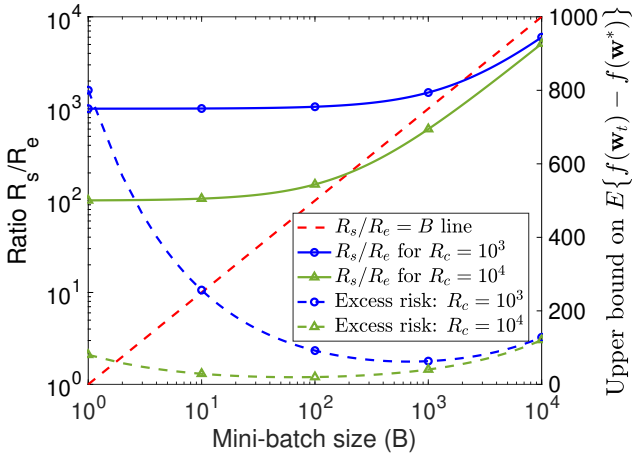


Fig. 5. An illustration of the impact of mini-batching on distributed, streaming processing under the exact averaging paradigm.

arriving at the system provided $R_s/R_e \leq B$, and we observe that for sufficiently large mini-batch size B, the ratio indeed drops below the $R_s/R_e = B$ line plotted in Fig. 5.

Next, we also overlay corresponding plots of the excess risk predicted for *Distributed Minibatch* SGD, presented in Section IV-A, after $t' = 10^6$ samples have arrived at the system. These plots show that increased mini-batch size helps the excess risk, but only to a point. Eventually, B becomes so large that the reduction in the number of algorithmic iterations carried out by the network hurts the overall performance more than the increase in the effective processing rate helps it. This illustrates that the mini-batch size B must be chosen judiciously, and in the following sections we will discuss theoretical results that shed light on this choice.

III. An Overview of the Technical Landscape

This paper ties together research in optimization and distributed processing within the context of machine learning. To elucidate the state of the art and set the stage for the results described in Sections IV and V, we present an overview of these areas and describe in detail key results that will be used later.

A. Optimization for Machine Learning

As mentioned in Section I-C, the literature on optimization for machine learning can be roughly divided into two interrelated frameworks, namely, *statistical risk minimization* (SRM) and *empirical risk minimization* (ERM). Both these frameworks aim to find a solution to the statistical optimization problem (2) and, as such, fall under the broad category of *stochastic optimization* (SO) within the optimization literature [38]. In particular, the SRM framework is often referred to as *stochastic approximation* (SA) and the ERM framework is sometimes termed *sample-average approximation* (SAA) in the literature [39]. In terms of specifics, the SA/SRM framework considers directly the statistical learning problem (2), and researchers have developed algorithms that minimize the risk $f(\mathbf{w})$ using "noisy" (stochastic) samples of its gradient $\nabla f(\mathbf{w})$. In contrast, the risk in the SAA/ERM framework is approximated by the empirical distribution over a fixed training dataset $\mathcal{Z}_T := \{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ of T data samples. This *empirical* risk,

defined as $\hat{f}(\mathbf{w}) := \frac{1}{T}\sum_{t'=1}^{T} \ell(\mathbf{w}, \mathbf{z}_{t'})$, is then minimized directly within the ERM framework, usually via some form of gradient-based (first-order) deterministic optimization methods. In the following, we describe a few key results from these two frameworks that are the most relevant to our discussion in this paper.

1) Stochastic Approximation (SA): The general assumption within the SA framework is that one has access to a stream of noisy gradients $\{\mathbf{g}_1, \mathbf{g}_2, \dots\}$ in order to solve (2), where the noisy gradient $\mathbf{g}_t$ at iteration t is defined as

$$\mathbf{g}_t := \nabla f(\mathbf{w}_t) + \boldsymbol{\zeta}_t, \tag{5}$$

with $\boldsymbol{\zeta}_t$ denoting i.i.d. noise with mean zero and finite variance, i.e., $\mathbb{E}\{\|\boldsymbol{\zeta}_t\|_2^2\} \leq \sigma^2$. In the parlance of SA, we have access to a first-order "oracle" that can be queried for a noisy gradient evaluated at the query point $\mathbf{w}_t$. In the parlance of machine learning, we have a stream of data samples $\{\mathbf{z}_1, \mathbf{z}_2, \dots\}$, each drawn i.i.d. according to the data distribution $\mathcal{D}$, and we solve (2) using the gradients $\mathbf{g}_t := \nabla \ell(\mathbf{w}_t, \mathbf{z}_t)$, which have gradient noise variance as defined in Definition 4.[2] It is straightforward to verify that these two formulations are equivalent: $\mathbb{E}\{\mathbf{g}_t\} = \nabla f(\mathbf{w}_t)$, so we can define $\boldsymbol{\zeta}_t := \mathbf{g}_t - \nabla f(\mathbf{w}_t)$ to be the zero-mean gradient noise in our problem setup.

The prototypical SA algorithm for loss functions whose gradients exist is *stochastic gradient descent* (SGD) [40], in which iterations/iterates take the form

$$\mathbf{w}_{t+1} = [\mathbf{w}_t - \eta_t \mathbf{g}_t]_{\mathcal{W}}\,, \tag{6}$$

where $[\cdot]_{\mathcal{W}}$ denotes projection onto the constraint set $\mathcal{W}$ and $\eta_t > 0$ denotes an appropriate *stepsize* that is either fixed (*constant* stepsize) or that decays to 0 with increasing t according to a prescribed strategy (*decaying* stepsize).

Remark 1. The term 'stochastic gradient descent' is overloaded in the literature. Many papers (e.g., [41], [42]) use the term in the SA sense described here, with a continuous stream of data in which no sample is used more than once. However, other papers (e.g., [2], [43]) use the term within the ERM framework to describe algorithms that operate on a fixed dataset, from which mini-batches of data are sampled with replacement and noisy gradients are computed. To disambiguate, some authors (e.g., [44]) use the term *single-pass* SGD to indicate the former usage.

Convex Problems. A common elaboration on SGD for convex loss functions is *Polyak–Ruppert averaging* [41], [45]–[47], in which a running average of iterates $\mathbf{w}_t$ is maintained as $\mathbf{w}_t^{\mathsf{av}} := \frac{1}{t}\sum_{\tau=0}^{t-1} \mathbf{w}_\tau$. The convergence rates of SGD for convex SA have been studied under a variety of settings, both with and without iterate averaging. The following result with a modified form of Polyak–Ruppert averaging comes from [3], in which iterate averaging takes the form

$$\mathbf{w}_t^{\mathsf{av}} := \left(\sum_{\tau=0}^{t-1} \eta_\tau\right)^{-1} \sum_{\tau=0}^{t-1} \eta_\tau \mathbf{w}_{\tau+1}. \tag{7}$$

[2]Note that the data arrival index t' and the algorithmic iteration index t are one and the same in a centralized setting; we are using t here in lieu of t' to facilitate comparisons with results in distributed settings.

Theorem 1. *For convex and smooth loss functions $\ell(\mathbf{w}, \mathbf{z})$ with (gradient) Lipschitz constant L, gradient noise variance σ^2, and bounded model space with expanse $D_{\mathcal{W}}$, there exist stepsizes η_t such that the approximation error of SGD with iterate averaging in* (7) *satisfies:*

$$\mathbb{E}\{f(\mathbf{w}_t^{\mathsf{av}})\} - f(\mathbf{w}^*) = O(1)\left[\frac{L}{t} + \frac{\sigma}{\sqrt{t}}\right]. \tag{8}$$

Remark 2. In [3], an optimal *constant* stepsize $\eta_t = \eta$ is given in the case where the optimization ends at a finite time horizon $t := T$ known in advance. In this case, the prescribed stepsize is $\eta := \min\left\{1/(2L), \sqrt{D_{\mathcal{W}}^2/(2T)}\right\}$, and this achieves the bound in (8). When the time horizon is unknown, a varying stepsize policy $\eta_t = O(1/\sqrt{t})$ achieves expected excess risk $O(\sigma/\sqrt{t})$, which is optimum for t much larger than L. For simplicity, we are working with the optimum stepsize proposed in [3] to retain the analysis for t not necessarily much larger than L.

Remark 3. It is desirable in some applications to state the SGD results in terms of convergence of the averaged iterate $\mathbf{w}_t^{\mathsf{av}}$ to $\mathbf{w}^*$. In the case of convex, smooth, and twice continuously differentiable loss functions, [47] provides such results for Polyak–Ruppert averaging in the almost sure sense and also proves asymptotic normality of $\mathbf{w}_t^{\mathsf{av}}$, i.e., $\sqrt{t}(\mathbf{w}_t^{\mathsf{av}} - \mathbf{w}^*)$ converges to a zero-mean Gaussian vector. In the case of strongly convex and smooth loss functions, [41] derives non-asymptotic convergence results for the Polyak–Ruppert averaged iterate $\mathbf{w}_t^{\mathsf{av}}$ in the mean-square sense. However, since machine learning is often concerned with minimizing the excess risk $\mathbb{E}\{f(\mathbf{w}_t^{\mathsf{av}}) - f(\mathbf{w}^*)\}$, we do not indulge further in discussion of convergence of the SGD iterates to the Bayes optimal solution $\mathbf{w}^*$.

A natural question is whether the convergence rate of Theorem 1 can be improved upon by another algorithm. It has been shown that incorporating *Nesterov's acceleration* [48] into SGD can indeed improve this rate somewhat. Roughly speaking, Nesterov's acceleration introduces a "momentum" term into the SGD iterations, allowing the directions of previous gradients to impact the direction taken during the current step and thereby speeding up convergence. The following formulation is an SGD-based simplification of the *accelerated stochastic mirror descent* algorithm of [3]. Define the *accelerated SGD* updates as follows:

$$\mathbf{u}_t = \beta_t^{-1}\mathbf{v}_t + (1 - \beta_t^{-1})\mathbf{w}_t, \tag{9}$$

$$\mathbf{v}_{t+1} = [\mathbf{u}_t - \eta_t \mathbf{g}_t]_{\mathcal{W}}\,, \text{ and} \tag{10}$$

$$\mathbf{w}_{t+1} = \beta_t^{-1}\mathbf{v}_{t+1} + (1 - \beta_t^{-1})\mathbf{w}_t, \tag{11}$$

where $\mathbf{g}_t := \nabla f(\mathbf{u}_t) + \boldsymbol{\zeta}_t$, and $\beta_t > 0$ and $\eta_t > 0$ are stepsizes. We then have the following result from [3].

Theorem 2. *For convex and smooth loss functions $\ell(\mathbf{w}, \mathbf{z})$ with (gradient) Lipschitz constant L, gradient noise variance σ^2, and bounded model space with expanse $D_{\mathcal{W}}$, there are stepsizes η_t and β_t such that the expected risk of accelerated SGD is bounded by*

$$\mathbb{E}\{f(\mathbf{w}_t)\} - f(\mathbf{w}^*) = O(1)\left[\frac{L}{t^2} + \frac{\sigma}{\sqrt{t}}\right]. \tag{12}$$

Remark 4. Similar to standard SGD, [3] prescribes stepsizes in the case of known and finite time horizon T, with $\beta_t = t/2$ and $\eta = t/2 \min\{1/(2L), \sqrt{6}/D_{\mathcal{W}}/(\sigma(T+1)^{3/2})\}$. Again a varying stepsize policy achieves excess risk

$O(\sigma/\sqrt{T})$ for large t, and we suppose the optimum stepsize given in [3] in order to facilitate analysis for t not necessarily much larger than L.

Both Theorem 1 and Theorem 2 explicitly bring out the dependence of the convergence rates on the gradient noise variance σ^2. In doing so, they hint at the potential performance advantages of (centralized or distributed) mini-batching of data. As the number of samples/iterations t goes to infinity and all else is held constant, the $O(\sigma/\sqrt{t})$ terms dominate the convergence rates in (8) and (12). Mini-batching can reduce the "equivalent" noise variance σ^2 of the mini-batched data samples and speed up convergence, but only to a point. If mini-batching forces the $O(\sigma/\sqrt{t})$ term smaller than the respective first terms in (8) and (12), then gradient noise is no longer the bottleneck to performance and mini-batching cannot improve convergence speed any further. Indeed, in the sequel we will choose the mini-batch size to carefully balance the two terms in (8) and (12), and we will further see that more aggressive mini-batching is advantageous when using accelerated methods.

We conclude our discussion of SA for convex loss functions by noting that the convergence rate of accelerated SGD is provably optimal for smooth, convex SA problems in the *minimax* sense: there is no single algorithm that can converge for all such SA problems at a rate faster than $O(L/t^2 + \sigma/\sqrt{t})$. (See [3] for an argument for this.) However, generalized and sometimes improved rates are possible outside of the regime of this setting. In particular, when $\ell(\mathbf{w}, \mathbf{z})$ is smooth and *strongly* convex, a convergence rate of $O(\sigma^2/t)$ is possible for σ^2 bounded away from zero, and it is the minimax rate [41], [49]. Results are also available when the loss function is non-smooth, when the solution is sparse or otherwise structured, and when the optimization space has a geometry that can be exploited to speed up convergence [3], [50], [51].

Nonconvex Problems. Nonconvex functions can have three types of critical points, defined as points $\mathbf{w}$ for which $\nabla f(\mathbf{w}) = \mathbf{0}$: *saddle points*, *local minima*, and *global minima*. This makes optimization of nonconvex (loss) functions using only first-order (gradient) information challenging. While works such as [52]–[58] provide convergence rates for nonconvex problems that are similar to their convex programming analogs, the convergence is only guaranteed to a critical point that is not necessarily a global optimum. Nonetheless, global optimization of nonconvex SA problems has been studied in the literature under a variety of assumptions on the geometry of objective functions. A major strand of work in this direction involves modifying the canonical SGD algorithm by injecting slowly decreasing Monte Carlo noise in its iterations. The resulting SA methods have been investigated in works such as [59]–[65] under the monikers of (continuous) *simulated annealing* and *stochastic gradient Langevin dynamics*. (Strictly speaking, [65] does not fall under the SA framework being discussed in this section.) A recent work [66] also provides global convergence guarantees for SGD for the class of (nonconvex) Morse functions.

Another major strand of work in global optimization of nonconvex functions involves explicit exploitation of the geometry of *structured* nonconvex problems such as *principal component analysis* (PCA), dictionary learning, phase retrieval, and low-rank matrix completion for global convergence guarantees. In this paper, we focus on one such structured nonconvex SA problem that corresponds to estimating the top eigenvector $\mathbf{w}^* \in \mathbb{R}^d$ of the covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$ of i.i.d. samples $\{\mathbf{z}_1, \mathbf{z}_2, \dots\} \subset \mathbb{R}^d$. The investigation of this 1-*PCA* problem in the paper, whose global convergence behavior has been investigated in works such as [42], [67]–[69], serves two purposes. First, it

helps validate the generality of the main message of this paper that the mismatches between the data streaming rate, compute rate, and communications rate can be accounted for through judicious choices of system parameters such as R, B, and N. Second, it helps crystallize the key characteristics of any global convergence analysis of nonconvex problems that can facilitate the convergence speed-up guarantees for the distributed mini-batch framework.

The loss function for the 1-PCA problem under the assumption of zero-mean distribution $\mathcal{D}$ supported on $\mathbb{R}^d$ and having covariance matrix $\mathbf{\Sigma} := \mathbb{E}_{\mathbf{z}\sim\mathcal{D}}\{\mathbf{z}\mathbf{z}^{\mathrm{T}}\}$ takes the form

$$\ell(\mathbf{w}, \mathbf{z}) = -\frac{\mathbf{w}^{\mathrm{T}}\mathbf{z}\mathbf{z}^{\mathrm{T}}\mathbf{w}}{\|\mathbf{w}\|_2^2}. \tag{13}$$

Note that $\nabla\ell(\mathbf{w}, \mathbf{z}) = -\frac{2\mathbf{z}\mathbf{z}^{\mathrm{T}}\mathbf{w}}{\|\mathbf{w}\|_2^2} + \frac{2(\mathbf{w}^{\mathrm{T}}\mathbf{z}\mathbf{z}^{\mathrm{T}}\mathbf{w})\mathbf{w}}{\|\mathbf{w}\|_2^4}$ and the optimal solution $\mathbf{w}^* \in \arg\min_{\mathbf{w}} [f(\mathbf{w}) := \mathbb{E}_{\mathbf{z}\in\mathcal{D}}\ell(\mathbf{w}, \mathbf{z})]$ corresponds to the dominant eigenvector of $\mathbf{\Sigma}$. In this paper, we focus on the SA approach termed Krasulina's method [70] that approximates the optimal solution $\mathbf{w}^*$ from data stream $\{\mathbf{z}_1, \mathbf{z}_2, \dots\}$ using iterations of the form

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t\Big(\mathbf{z}_t\mathbf{z}_t^{\mathrm{T}}\mathbf{w}_{t-1} - \frac{\mathbf{w}_{t-1}^{\mathrm{T}}\mathbf{z}_t\mathbf{z}_t^{\mathrm{T}}\mathbf{w}_{t-1}}{\|\mathbf{w}_{t-1}\|_2^2}\,\mathbf{w}_{t-1}\Big). \tag{14}$$

Notice that changing η_t to $\frac{\eta_t}{\|\mathbf{w}_{t-1}\|_2^2}$ in (14) gives us the SGD iteration. Despite the empirical success of SA iterations such as (14) in approximating the top eigenvector of $\mathbf{\Sigma}$, earlier works only provided asymptotic convergence guarantees for such methods. Recent studies such as [42], [67], [68], [71], [72] have filled this gap by providing non-asymptotic results. The following theorem, which is due to [67], provides guarantees for Krasulina's method.

Theorem 3. *Let the i.i.d. data samples be bounded, i.e.,* $\forall t, \|\mathbf{z}_t\|_2 \leq \kappa$*, define* $\mathtt{gap} := \lambda_1(\mathbf{\Sigma}) - \lambda_2(\mathbf{\Sigma}) > 0$*, fix any* $\delta \in (0, 1)$*, and define* $c := \frac{c_0}{2\mathtt{gap}}$ *for any* $c_0 > 2$*. Next, pick any*

$$Q \geq \frac{512e^2d^2\kappa^4\max(1, c^2)}{\delta^4}\ln\frac{4}{\delta} \tag{15}$$

and choose the stepsize sequence as $\eta_t := c/(Q + t)$*. Then there exists a sequence* $(\Omega_t^{'})_{t\in\mathbb{Z}_+}$ *of nested subsets of the sample space* Ω *such that* $\mathbb{P}\left(\cap_{t>0}\Omega_t^{'}\right) \geq 1 - \delta$ *and*

$$\mathbb{E}_t\big\{f(\mathbf{w}_t)\big\} - f(\mathbf{w}^*) \leq C_1\Big(\frac{Q+1}{t+Q+1}\Big)^{\frac{c_0}{2}} + C_2\Big(\frac{\kappa^2}{t+Q+1}\Big), \tag{16}$$

where $\mathbb{E}_t$ *is the conditional expectation over* $\Omega_t^{'}$*, and* C_1 *and* C_2 *are constants defined as*

$$C_1 := \frac{\lambda_1(\mathbf{\Sigma})}{2}\left(\frac{4ed}{\delta^2}\right)^{\frac{5}{2\ln 2}} e^{2c^2\lambda_1^2(\mathbf{\Sigma})/Q} \quad \text{and} \quad C_2 := \frac{2c^2\lambda_1(\mathbf{\Sigma})e^{(c_0+2c^2\lambda_1^2(\mathbf{\Sigma}))/Q}}{(c_0-2)}.$$

The convergence guarantees in Theorem 3 depend on problem parameters such as d, $\mathtt{gap}$, and δ. Recent works [71], [73] have provided lower bounds on the dependence of convergence rates on these parameters for the stochastic PCA problem. Theorem 3 achieves these lower bounds with respect to $\mathtt{gap}$ and δ up to logarithmic factors. But the dependence on data dimension in Theorem 3 is d^4, while the lower bound suggests $\Omega(\log(d))$ dependence. In addition, convergence guarantees for a variant of Krasulina's method termed Oja's algorithm are known to achieve this lower bound dependence on data dimensionality [68], [71], [72], [74].

Despite this somewhat suboptimal nature of Theorem 3, Krasulina's method lends itself to relatively simpler analysis for the distributed (mini-batch) framework being studied in this paper. Specifically, as alluded to in our discussion in Section II-A, implicit averaging out of the sample-covariance noise is the key reason for the potential

speed-up in convergence within any distributed processing framework. And while Theorem 3 does not have an explicit dependence on the noise variance σ^2, a variance-based analysis of Krasulina's method—discussed in detail in Section IV and having similar dependence on d, gap, and δ as Theorem 3—has been provided in a recent work [75]. In contrast, results in [71], [72] are oblivious to the variance in sample covariance and hence cannot be used to show faster convergence within distributed frameworks. On the other hand, while the results in [68], [74] do take the noise variance into account, the probability of success in these works cannot be improved beyond $3/4$ in a single-pass SA setting.

2) Empirical Risk Minimization (ERM): Given the *fixed* training dataset $\mathcal{Z}_T$ of T i.i.d. samples drawn from the distribution $\mathcal{D}$ and the corresponding empirical risk $\hat{f}(\mathbf{w})$, the main objective within the ERM framework is to directly minimize $\hat{f}(\mathbf{w})$ in order to obtain the ERM solution $\mathbf{w}^*_{\text{ERM}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \hat{f}(\mathbf{w})$. Such problems, sometimes referred to as *finite-sum optimization problems*, have traditionally been solved using (deterministic, projected) gradient descent or similar methods. But the advent of massive datasets has made direct computations of gradients of $\hat{f}(\mathbf{w})$ intractable. This has led to the development of several families of SGD-type methods for the ERM problem, where the stochasiticity in these methods refers to noisy gradients of the empirical risk $\hat{f}(\mathbf{w})$, as opposed to noisy gradients of the true risk $f(\mathbf{w})$ within the (single-pass) SA framework. Specifically, the prototypical SGD algorithm for the ERM problem samples with replacement a single data sample $\mathbf{z}_k$ (or a small mini-batch of samples) from $\mathcal{Z}_T$ in each iteration k, computes the gradient $\nabla\ell(\mathbf{w}_k, \mathbf{z}_k)$, and takes a step in the negative of the computed gradient's direction. The iterates $\mathbf{w}_k$ of this particular SGD variant are known to converge reasonably fast to the ERM solution $\mathbf{w}^*_{\text{ERM}}$ under various assumptions on the geometry of the loss function $\ell(\mathbf{w}, \mathbf{z})$ [2], [76].

A variety of adaptive and more elaborate SGD-style algorithms, such as Adagrad, RMSProp, and Adam [77], [78], which introduce adaptive stepsizes, momentum terms, and Nesterov-style acceleration, have been developed in recent years. Empirically, these methods provide faster convergence to at least a stationary point of $\hat{f}(\mathbf{w})$, especially when training deep neural networks. (Note that some of these methods have provable convergence issues, even for convex problems [79].) A family of so-called *variance-reduction* methods [4], [69], [80]–[82], such as *stochastic variance reduced gradient* (SVRG), *stochastically controlled stochastic gradient* (SCSG), and NATASHA, have also been developed in the literature for the ERM problem. In these methods, iterates from previous epochs are averaged to produce a low-complexity estimate of the gradient with provably small variance, which speeds up convergence. In terms of theoretical analysis, SGD-style and variance-reduction algorithms are studied in both convex and nonconvex settings. Unlike the SA framework, however, the convergence analysis of these methods for the ERM setting is in terms of the computational effort, measured in terms of the number of gradient evaluations, needed to approach a global optimum or a stationary point of the empirical risk $\hat{f}(\mathbf{w})$.

Since optimization methods for the ERM framework primarily provide bounds on either $[\hat{f}(\mathbf{w}_k) - \hat{f}(\mathbf{w}^*_{\text{ERM}})]$ or $\|\mathbf{w}_k - \mathbf{w}^*_{\text{ERM}}\|_2$, a bound on the excess risk $[f(\mathbf{w}_k) - f(\mathbf{w}^*)]$ under the ERM setting necessitates additional analytical steps that typically involve bounding the *generalization error*, defined as $[f(\mathbf{w}^*_{\text{ERM}}) - \hat{f}(\mathbf{w}^*_{\text{ERM}})]$, of the ERM solution. Classic generalization error bounds have been provided in terms of the *Vapnik–Chervonenkis dimension* or *Rademacher complexity* of the class of functions induced by $\mathcal{W}$ [34], [83], or in terms of the uniform or so-called "leave-one-out" stability [84]–[87] of the solution. Together, the optimization-theoretic bounds and the *learning-*

theoretic bounds on quantities such as the generalization error result in excess risk bounds that decay at rates $O(T^{-1/2})$ or $O(T^{-1})$ for various loss functions as long as the number of optimization iterations k is on the order of the number of training samples T. Thus, the ERM framework can yield excess risk bounds that match the sample complexity of the ones under the SA framework. Nonetheless, we focus primarily on the SA setting in this paper for two reasons. First, we are concerned with the statistical optimization problem (2), and the SA framework measures performance directly with respect to this problem, whereas the ERM/finite-sum setting yields the final results only after a combination of optimization-theoretic and learning-theoretic bounds. Second, the SA framework is naturally well-suited to the setting of streaming data, whereas ERM supposes access to the entire dataset.

B. Distributed Optimization and Machine Learning

Distributed optimization is an extremely broad field, with a rich history. In this paper, we only discuss the portion of the literature most relevant to our problem setting. Specifically, we focus on methods for distributing SGD-style algorithms over collections of computing devices and/or processors that communicate over networks defined by graphs and aggregate data by *averaging* information over the network. We further divide these methods into two categories, based on the nature of distributed averaging that is employed within each algorithm: *exact averaging*, in which processing nodes use a robust *message passing interface* (MPI) communications primitive such as `AllReduce` [27] to compute exact averages of gradients and/or iterates in the network, and *inexact averaging*, in which an approximate approach such as distributed consensus/diffusion [8]–[10] is used to *approximate* averages of gradients and/or iterates in the network. The former category of algorithms requires careful network configuration in order to coordinate `AllReduce`-style averaging, whereas the latter category requires minimal explicit configuration, but the algorithms can suffer from slower convergence due to approximation error in the averaging step.

1) Exact Averaging and Distributed Machine Learning: In the case of algorithms utilizing exact averaging, processing nodes employ an MPI library to compute exact averages in a robust manner. While implementations differ, a generic approach is to compute averages over a spanning tree in the network. Reusing the notation introduced in Section II-C, let $\{\mathbf{v}_n \in \mathbb{R}^d\}_{n\in\mathcal{V}}$ be the set of vectors distributed across the network at the start of the averaging subroutine and let $\bar{\mathbf{v}} := \frac{1}{N}\sum_{n\in\mathcal{V}} \mathbf{v}_n$ denote their average. Then, the average $\bar{\mathbf{v}}$ can be obtained at each node in a two-pass manner. In the first pass, each leaf node n in the spanning tree passes its vector $\mathbf{v}_n$ to its parent node, which averages together the vectors of its child nodes and passes the average to *its* parent node; this process continues recursively until the root node has the average $\bar{\mathbf{v}}$. In the second pass, the root node disseminates $\bar{\mathbf{v}}$ to the network by passing it to its child nodes; this continues recursively until all of the leaf nodes posses $\bar{\mathbf{v}}$. This type of averaging is provably efficient, requiring only $R = O(N)$ exchange of messages within the network.

This generic approach to computing exact averages has been applied to distributed machine learning via a variety of implementations, especially under the distributed computing framework. `TensorFlow` has a package for parameter-server distributed learning on multiple GPUs that uses exact averaging; worker nodes compute gradients, which are forwarded to the parameter server for exact averaging [25]. By contrast, `Horovod` [88] is a distributed-parameter library for deep learning that averages gradients using *ring* `AllReduce`; the GPU nodes are connected into a ring topology, which makes for simple and efficient exact averaging.

2) Inexact Averaging and Distributed Machine Learning: In the case of algorithms utilizing inexact averaging, processing nodes use local communications, without network-wide coordination, to compute approximate averages of their data. A widespread method for this is *averaging consensus*, a mainstay of distributed control, signal processing, and learning [15], [89]. Again suppose $\{\mathbf{v}_n \in \mathbb{R}^d\}_{n\in\mathcal{V}}$ is the set of vectors distributed across the network at the start of the averaging subroutine and $\bar{\mathbf{v}}$ denotes the exact average of these vectors. Next, define a *doubly stochastic matrix* $\mathbf{A} \in \mathbb{R}^{N\times N}$ that is *consistent* with the topology of the network $G = (\mathcal{V}, \mathcal{E})$. That is, $\mathbf{A}$ is a matrix whose entries are non-negative, whose rows and columns sum to one, whose diagonal entries $a_{n,n}$ are non-zero, and whose (n, m)-th entry $a_{n,m} \neq 0$ only when $(n, m) \in \mathcal{E}$. Averaging consensus then proceeds in multiple rounds of the following iteration using local communications:

$$\mathbf{v}_n^{r+1} = \sum_{m=1}^{N} a_{n,m}\mathbf{v}_m^r. \tag{17}$$

Here, $r \in \mathbb{Z}_+$ denotes the iteration index for averaging consensus, $\mathbf{v}_n^r$ denotes an approximation of $\bar{\mathbf{v}}$ at node n after r iterations, and $\mathbf{v}_n^0 := \mathbf{v}_n$. In words, each processing node takes a convex combination of the estimate of $\bar{\mathbf{v}}$ at its neighboring nodes. Under mild conditions, averaging consensus converges geometrically on $\bar{\mathbf{v}}$, with the approximation error scaling as $\|\mathbf{v}_n^r - \bar{\mathbf{v}}\|_2 = O\left(|\lambda_2(\mathbf{A})|^r\right)$.

Distributed gradient descent (DGD) is a classic approach to distributed optimization via inexact averaging [12]. It uses only a single round of averaging consensus per iteration, i.e., $R = 1$ using the notation of Section II-C, and it is posed in the setting of finite-sum optimization: each node n has a local cost function $\hat{f}_n(\mathbf{w})$, and the objective is to minimize the sum $\hat{f}(\mathbf{w}) := \sum_{n=1}^{N} \hat{f}_n(\mathbf{w})$. While DGD was originally posed in the framework of distributed control, it applies equally well to the distributed ERM setting in which $\hat{f}_n(\mathbf{w})$ corresponds to the empirical risk over the training data at node n. In terms of specifics, the original DGD formulation supposes a synchronous communications model in which each node n computes a weighted average of its neighbors' iterates at each iteration t, after which it takes a gradient step with respect to its local cost function:

$$\mathbf{w}_{n,t+1} = \mathbf{w}_{n,t} + \sum_{m=1}^{N} a_{n,m}\mathbf{w}_{m,t} - \eta_t \nabla \hat{f}_n(\mathbf{w}_{n,t}). \tag{18}$$

Thus, each node takes a standard gradient descent step preceded by one-round averaging consensus on the iterates.

Several extensions to DGD have been proposed in the literature, including extensions to time-varying and directed graphs [90]–[92] and variations with stronger convergence guarantees [93]–[95]. Other related works have studied distributed (stochastic) optimization via means other than gradient descent, including distributed dual averaging [96], [97] and the *alternating direction method of multipliers* (ADMM) [98]–[100]. The convergence of DGD-style methods has been studied under a variety of settings; two relevant results are that stochastic DGD-style algorithms have error decaying as $O(\log(t)/\sqrt{t})$ for general smooth convex functions and $O(\log(t)/t)$ for smooth strongly convex functions, even if the network is time varying [90], [91].

We conclude by noting that inexact averaging-based distributed algorithms have also been analyzed/proposed for nonconvex optimization problems. In particular, DGD-style methods for nonconvex finite-sum problems are presented in [101]–[103], and convergence rates to stationary points and, when possible, local minima are derived.

Further particularization of these works to problems with "nicer" geometry of saddle points and to structured nonconvex problems such as PCA can be found in works such as [104]–[107].

C. Roadmap for the Remainder of the Paper

Putting the results presented in this paper in the context of the preceding discussion, the rest of the paper describes recent results in distributed machine learning from fast streaming data over networks that aggregate the distributed information using both exact and inexact averaging. Specifically, we synthesize results from four recent papers [75], [108]–[110] that focus on the distributed SA setting of Section II. Among these works, nodes in [75], [108] exchange messages using a robust MPI primitive such as `AllReduce`, allowing exact averaging of messages for processing. The main distinction between these two works is that [108] focuses on distributed convex SA problems, whereas [75] studies the distributed PCA problem under the SA setting. In contrast, nodes in [109], [110] exchange messages using multiple rounds of averaging consensus and thus, similar to DGD, aggregate information using inexact averaging of messages. Both these works study distributed convex SA problems, with [109] focusing on dual averaging and [110] investigating gradient descent as solution strategies.

IV. Distributed Stochastic Approximation Using Exact Averaging

We detail two machine learning algorithms in this section for the distributed mini-batch framework of Section II, with one algorithm for general convex loss functions and the other one for the nonconvex loss function corresponding to the 1-PCA problem. Both these algorithms operate under the assumption of nodes aggregating distributed information via exact averaging using `AllReduce`-style communications primitives. The main focus in both these algorithms is to strike a balance between streaming, computing, and communications rates, while ensuring that the error in the final estimates is near optimal in terms of the number of samples arriving at the distributed system.

Both the algorithms take advantage of the fact that (implicit or explicit) mini-batching reduces (gradient / sample covariance) noise variance. Between any two data-splitting instances, nodes in each algorithm compute average gradients/iterates over the newest (network-wide) B data samples and use these *exactly* averaged quantities for a stochastic update. Given ample compute resources and keeping everything else fixed, an increase in network-wide mini-batch size B under such a strategy decreases both the noise variance and demands on the communications resources. In doing so, however, one also reduces the number of algorithmic iterations that take place within the network per second, which has the potential to slow down the convergence rates of the algorithms to the optimal solutions. An important question then is whether (and when) it is possible to utilize network-wide mini-batch averaging to simultaneously balance the compute-limited and communications-limited regimes in high-rate streaming settings (i.e., ensure $\frac{R_s}{R_e} \not\gg B$), reduce the noise variance, and guarantee that (order-wise) the convergence rate is not adversely impacted. We address this question in the following for the case of exact averaging.

A. Distributed Mini-batched Stochastic Convex Approximation

Due to the high impact of mini-batching on the performance of distributed stochastic optimization, distributed methods deploying mini-batching and utilizing exact averaging have been studied extensively in the past few years;

Algorithm 1 The Distributed Mini-batch (DMB) Algorithm [108]

Require: Provisioning of compute and communications resources to ensure fast effective processing rate, i.e., either $R_s \leq BR_e$ or $R_s = (B+\mu)\,R_e$, as well as guaranteed exact averaging in R rounds of communications

Input: Data stream $\{\mathbf{z}_{t'} \overset{\text{i.i.d.}}{\sim} \mathcal{D}\}_{t' \in \mathbb{Z}_+}$ that is split into N streams of mini-batched data $\{\mathbf{z}_{n,b,t}\}_{b=1,t\in\mathbb{Z}_+}^{B/N}$ across the network of N nodes (after possible discarding of μ samples per split) and stepsize sequence $\{\eta_t \in \mathbb{R}_+\}_{t\in\mathbb{Z}_+}$

Initialize: All compute nodes initialize with $\mathbf{w}_0 = \mathbf{0} \in \mathbb{R}^d$

1: **for** $t = 1, 2, \ldots,$ **do**
2: $\quad \forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{0} \in \mathbb{R}^d$
3: $\quad$ **for** $b = 1, \ldots, B/N$ **do** $\quad \triangleright$ Node n receives the mini-batch $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ and updates $\mathbf{g}_{n,t}$ locally
4: $\quad\quad \forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,b,t} \leftarrow \nabla \ell(\mathbf{w}_t, \mathbf{z}_{n,b,t})$
5: $\quad\quad \forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{g}_{n,t} + \frac{1}{B/N}\mathbf{g}_{n,b,t}$
6: $\quad$ **end for**
7: $\quad$ Compute $\mathbf{g}_t \leftarrow \frac{1}{N}\sum_{n=1}^{N} \mathbf{g}_{n,t}$ in the network using exact averaging
8: $\quad$ Set $\mathbf{w}_{t+1} \leftarrow [\mathbf{w}_t - \eta_t \mathbf{g}_t]_{\mathcal{W}}$ across the network
9: $\quad$ **if** $R_s = (B+\mu)\,R_e$ **then** $\quad \triangleright$ Slight under-provisioning of compute/communications resources
10: $\quad\quad$ The system receives $(B+\mu)$ additional data samples during execution of Steps 2–8, out of which $\mu \in \mathbb{Z}_+$ samples are discarded at the splitter
11: $\quad$ **end if**
12: **end for**

Return: An estimate $\mathbf{w}_t$ of the Bayes optimal solution after receiving $t' = (B+\mu)t$ samples

see, e.g., [5], [108], [111], [112]. Among these works, the results in [108] provide an upper bound on the network-wide mini-batch size B that ensures sample-wise order-optimal convergence in SA settings. In contrast, [5], [111], [112] focus on the selection of mini-batch size under ERM settings. Since the SA setting is best suited for the streaming framework of this paper, our discussion here focuses exclusively on the *distributed mini-batch* (DMB) algorithm proposed in [108] for stochastic convex approximation. The DMB algorithm is listed as Algorithm 1 in the following and discussed further below.

We begin with the data-splitting model of Section II and initially assume sufficient provisioning of resources so that $R_s \leq BR_e$. The DMB algorithm at iteration t in this setting has a mini-batch $\{\mathbf{z}_{t'}, t' = (t-1)B+1, \ldots, tB\}$ of B data samples at the splitter, which is then distributed as N smaller mini-batches of size B/N each across the network of N compute nodes. Afterwards, the nodes in the network locally (and in parallel) compute an average gradient $\mathbf{g}_{n,t}$ of the loss function over their local mini-batch of B/N data samples (see Steps 3–6 in Algorithm 1). Next, nodes engage in distributed exact averaging of their local mini-batched gradients using an `AllReduce`-style communications primitive to obtain the network-wide mini-batched average gradient $\mathbf{g}_t$ (cf. Step 7, Algorithm 1), which is then used to update the network-wide estimate $\mathbf{w}_t$ of the machine learning model (cf. Step 8, Algorithm 1).

The DMB algorithm can also deal with *reasonable* under-provisioning of resources without sacrificing too much

in terms of the quality of the estimate $\mathbf{w}_t$. Recall that the distributed processing framework cannot process all incoming samples when $R_s > BR_e$. However, as long as $R_s \not\gg BR_e$, the DMB algorithm simply resorts to dropping μ $(\in \mathbb{Z}_+) := (\frac{R_s}{R_e} - B)$ samples per splitting instance at the splitter in this resource-constrained setting and then proceeds with Steps 2–8 using the remaining B samples as before.

The main analytical contribution of [108] was providing upper bounds on the mini-batch size B and, when necessary, the number of discarded samples μ that ensure sample-wise order-optimal convergence for the DMB algorithm. We summarize these results of [108] in the following theorem.

Theorem 4. *Let the loss function $\ell(\mathbf{w}, \mathbf{z})$ be convex and smooth with L-Lipschitz gradients and gradient noise variance σ^2. Then, assuming bounded model space $\mathcal{W}$ and choosing stepsizes as $\eta_t = \frac{1}{L+(\sigma/D_{\mathcal{W}})\sqrt{t}}$, the approximation error of Algorithm 1 after t iterations is bounded as follows:*

$$\mathbb{E}\left\{f(\mathbf{w}_t)\right\} - f(\mathbf{w}^*) \leq (B+\mu)\left(\frac{2D_{\mathcal{W}}^2 L}{t'} + \frac{2D_{\mathcal{W}}\sigma}{\sqrt{t'}}\right). \tag{19}$$

Furthermore, if $B = (t')^{\rho}$ for any $\rho \in (0, 1/2)$ and $\mu = o(B)$, then the approximation error is bounded as

$$\mathbb{E}\left\{f(\mathbf{w}_t)\right\} - f(\mathbf{w}^*) \leq \frac{2D_{\mathcal{W}}\sigma}{\sqrt{t'}} + o\Big(\frac{1}{\sqrt{t'}}\Big). \tag{20}$$

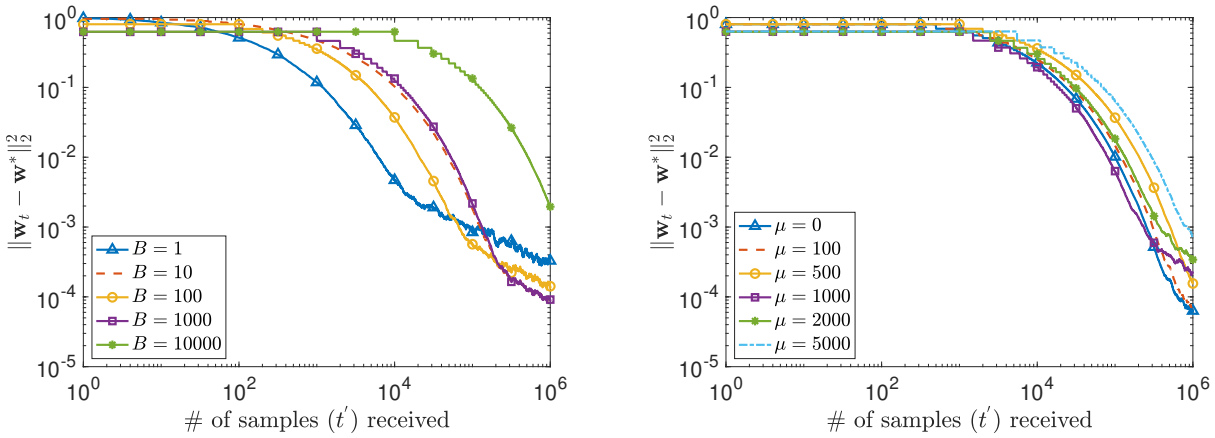
It can be seen from Theorem 4 that the DMB algorithm results in near-optimal convergence rate of $O(1/\sqrt{t'})$, which corresponds to speed-up by a factor of $O(B)$, in two cases. First, when $R_s \leq BR_e$ and thus $\mu \equiv 0$, it can be seen from (19) that this speed-up is obtained as long as $B = O(\sqrt{t'})$. Second, even when $R_s > BR_e$ and therefore $\mu = (\frac{R_s}{R_e} - B) > 0$, (19) guarantees the convergence speed-up as long as $B = o(\sqrt{t'})$ and $\mu = o(B)$. Stated differently, the speed-up can be obtained provided the streaming rate (R_s) does not exceed the effective processing rate per sample (BR_e) by too much, i.e., $\frac{R_s}{BR_e} = O(1)$.

B. Numerical Experiments for the DMB Algorithm

We demonstrate the effectiveness of the scaling laws implied by Theorem 4 by using the DMB algorithm to train a binary linear classifier (supervised learning problem) from streaming (labeled) data using *logistic regression* [113]. To this end, we take the labeled data as the tuple $\mathbf{z} := (\mathbf{x}, y)$ with $\mathbf{x} \in \mathbb{R}^d$ and the labels $y \in \{-1, 1\}$, define the regression model as $\mathbf{w} := (\widetilde{\mathbf{w}}, w_0) \in \mathbb{R}^d \times \mathbb{R}$, and recall that the convex and smooth loss function for logistic regression can be expressed as $\ell(\mathbf{w}, \mathbf{z}) = \ln\left(1 + \exp(-y(\widetilde{\mathbf{w}}^{\mathrm{T}}\mathbf{x} + w_0))\right)$. Note that the optimal batch solution for logistic regression corresponds to the maximum likelihood estimate of the ground-truth regression coefficients that generate data $\mathbf{z} = (\mathbf{x}, y)$ [113].

The experimental results reported in this section correspond to $d = 5$ and are averaged over 50 Monte Carlo trials. In order to generate data for each trial, we first generate ground-truth regression parameters via a random draw from the standard normal distribution, $\mathbf{w}^* = (\widetilde{\mathbf{w}}^*, w_0^*) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Next, we generate data samples as independent draws from another standard normal distribution, $\mathbf{x}_{t'} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and generate the corresponding labels $y_{t'}$ as independent draws from the Bernoulli distribution induced by the regression coefficients, i.e.,

$$\Pr(y_{t'} = 1 | \mathbf{x}_{t'}) = 1/(1 + \exp\left(-(\widetilde{\mathbf{w}}^{*\mathrm{T}}\mathbf{x}_{t'} + w_0^*)\right)). \tag{21}$$



(a) Impact of the mini-batch size on the convergence rate of the DMB algorithm for the resourceful regime. Note that the $B = 1$ plot is effectively standard SGD.

(b) Performance of the DMB algorithm in a resource-constrained regime (i.e., $R_s > BR_e$), which causes loss of μ samples per iteration; here, $(N, B) = (10, 500)$.

Fig. 6. Convergence behavior of the DMB algorithm for the case of synthetic data under two scenarios: (a) No data loss ($\mu = 0$) and (b) loss of $\mu > 0$ samples per algorithmic iteration.

We report results of two experiments for the distributed, streaming framework of Section II. The first experiment deals with the resourceful regime, i.e., $R_s \leq BR_e$, and uses mini-batches of size $B \in \{1, 10, 100, 1000\}$. The results, shown in Fig. 6(a), are obtained for stepsize of the form $c/\sqrt{t}$ (as prescribed by Theorem 4), where the corresponding value of c chosen for different batch sizes is $c \in \{0.1, 0.1, 0.5, 1, 1\}$. In order to select these values of c, we ran the experiment for multiple choices of c and picked the values that achieved the best results. Note that the results in Fig. 6(a) correspond to the optimality gap $\|\mathbf{w}_t - \mathbf{w}^*\|_2^2$ of the iterates from the ground truth; since the logistic loss is Lipschitz continuous, this trivially upper bounds the *square* of the excess risk $f(\mathbf{w}_t) - f(\mathbf{w}^*)$. It can be seen from these results that, as predicted by Theorem 4, the estimation error $\|\mathbf{w}_t - \mathbf{w}^*\|_2^2$ after $t = t'/B$ iterations of the DMB algorithm is roughly on the order of $O(1/t')$ for $B \in \{1, 10, 100, 1000\}$, while it is worse by an order of magnitude for $B = 10^4 > \sqrt{t'}$.

Next, we demonstrate the performance of the DMB algorithm for resource-constrained settings, i.e., $R_s > BR_e$, which causes the algorithm to discard $\mu = (R_s/R_e - B)$ samples per iteration. The experiment for this setting corresponds to a network of 10 nodes ($N = 10$) with network-wide mini-batch of size $B = 500$ (i.e., $B/N = 50$). We consider different mismatch factors between streaming, processing, and communication rates in this experiment, which result in the number of samples being discarded as $\mu \in \{0, 100, 500, 1000, 2000, 5000\}$. The results are plotted in Fig. 6(b), which shows that the error $\|\mathbf{w}_t - \mathbf{w}^*\|_2^2$ for $\mu = 100$ is comparable to that for $\mu = 0$ and progressively worsens as μ increases from $\mu = 500$ to $\mu = 5000$.

C. Distributed Mini-batched Streaming PCA

Mini-batching and variance-reduction techniques have also been utilized for nonconvex stochastic optimization problems in both centralized and distributed settings [23], [57], [69], [114]–[119]. But these and similar works

Algorithm 2 Distributed Mini-batch Krasulina (DM-Krasulina) Algorithm [75]

Require: Same as the DMB algorithm in Algorithm 1

Input: Same as the DMB algorithm in Algorithm 1

Initialize: All compute nodes initialize with the same $\mathbf{w}_0 \in \mathbb{R}^d$ randomly generated over the unit sphere

1: **for** $t = 1, 2, \ldots,$ **do**

2: $\quad \forall n \in \{1, \ldots, N\},\ \boldsymbol{\xi}_{n,t} \leftarrow \mathbf{0} \in \mathbb{R}^d$

3: $\quad$ **for** $b = 1, \ldots, B/N$ **do** $\quad \triangleright$ Node n receives the mini-batch $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ and updates $\boldsymbol{\xi}_{n,t}$ locally

4: $\quad\quad \forall n \in \{1, \ldots, N\},\ \boldsymbol{\xi}_{n,t} \leftarrow \boldsymbol{\xi}_{n,t} + \mathbf{z}_{n,b,t}\mathbf{z}_{n,b,t}^{\mathrm{T}}\mathbf{w}_{t-1} - \frac{\mathbf{w}_{t-1}^{\mathrm{T}}\mathbf{z}_{n,b,t}\mathbf{z}_{n,b,t}^{\mathrm{T}}\mathbf{w}_{t-1}\mathbf{w}_{t-1}}{\|\mathbf{w}_{t-1}\|_2^2}$

5: $\quad$ **end for**

6: $\quad$ Compute $\boldsymbol{\xi}_t \leftarrow \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\xi}_{n,t}$ in the network using exact averaging

7: $\quad$ Update the eigenvector estimate in the network as follows: $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + \eta_t\boldsymbol{\xi}_t$

8: $\quad$ **if** $R_s = (B + \mu)\, R_e$ **then** $\quad \triangleright$ Slight under-provisioning of compute/communications resources

9: $\quad\quad$ The system receives $(B + \mu)$ additional data samples during execution of Steps 2–7, out of which $\mu \in \mathbb{Z}_+$ samples are discarded at the splitter

10: $\quad$ **end if**

11: **end for**

Return: An estimate $\mathbf{w}_t$ of the eigenvector $\mathbf{w}^*$ associated with $\lambda_1(\boldsymbol{\Sigma})$ after receiving $t' = (B + \mu)t$ samples

typically either only guarantee convergence to first-order stationary points [69], [119] and/or they are not applicable to the single-pass SA setting [57], [114]–[118]. In the following, we focus on the structured nonconvex SA problem of estimating the top eigenvector of a covariance matrix from fast streaming i.i.d. data samples (i.e., the streaming 1-PCA problem). Global convergence guarantees for this problem, as noted in Section III-A, have been derived in the literature for "slow" data streams. Our discussion here revolves around the *distributed mini-batch Krasulina* (DM-Krasulina) algorithm that has been recently proposed and analyzed in [75] for the distributed mini-batch framework of Section II for fast streaming data.

The DM-Krasulina algorithm (see Algorithm 2) can be seen as a slight variation on the DMB algorithm for solving the 1-PCA problem from fast streaming data. In particular, DM-Krasulina is nearly identical to the DMB algorithm except for the fact that the generic gradients $\mathbf{g}_{n,t}$ and $\mathbf{g}_t$ in Algorithm 1 are replaced by pseudo-gradient terms $\boldsymbol{\xi}_{n,t}$ and $\boldsymbol{\xi}_t$, respectively, in Algorithm 2. Therefore, the implementation details provided for the DMB algorithm in Section IV-A also apply to DM-Krasulina. Nonetheless, the analytical tools utilized by [75] to theoretically characterize the interplay between the solution accuracy of DM-Krasulina and different system parameters differ greatly from those utilized for Theorem 4.

In order to discuss the convergence behavior of DM-Krasulina, we recall the assumptions stated in Section III-A for the 1-PCA problem. Specifically, the i.i.d. data samples $\mathbf{z}_{t'}$ have zero mean and are bounded almost surely by some positive constant κ, i.e., $\mathbb{E}\{\mathbf{z}_{t'}\} = \mathbf{0}$ and $\forall t', \|\mathbf{z}_{t'}\|_2 \leq \kappa$. Notice that Steps 3–6 in Algorithm 2 lead to an implicit computation of an unbiased estimate of the population covariance matrix $\boldsymbol{\Sigma} = \mathbb{E}_{\mathbf{z}\sim\mathcal{D}}\{\mathbf{z}\mathbf{z}^{\mathrm{T}}\}$ from the

network-wide mini-batch of B samples, which we denote by $\mathbf{A}_t := (1/B)\sum_{n=1}^{N}\sum_{b=1}^{B/N}\mathbf{z}_{n,b,t}\mathbf{z}_{n,b,t}^{\mathrm{T}}$. The results for DM-Krasulina depend on the variance of this unbiased sample covariance, which is defined as follows.

Definition 7 (Variance of sample covariance in DM-Krasulina)**.** The variance of the distributed sample covariance matrix $\mathbf{A}_t$ in DM-Krasulina is defined as follows:

$$\sigma_B^2 := \mathbb{E}_{\mathcal{D}}\left\{\left\|\frac{1}{B}\sum_{n=1}^{N}\sum_{b=1}^{B/N}\mathbf{z}_{n,b,t}\mathbf{z}_{n,b,t}^{\mathrm{T}} - \mathbf{\Sigma}\right\|_F^2\right\}.$$

Note that σ_B^2 for $B = 1$ corresponds to the single-sample covariance noise variance σ^2 defined in Section II. It is also straightforward to show that $\sigma_B^2 \leq \sigma^2/B$. And since all moments of the probability distribution $\mathcal{D}$ exist by virtue of the norm boundedness of $\mathbf{z}_{t'}$, the variance σ_B^2 as defined above exists and is finite. We now provide the main result for DM-Krasulina from [75] that expresses the convergence behavior of DM-Krasulina in terms of the mini-batched noise variance σ_B^2.

Theorem 5. *Let the i.i.d. data samples be bounded, i.e., $\forall t', \|\mathbf{z}_{t'}\|_2 \leq \kappa$, define $\mathtt{gap} := \lambda_1(\mathbf{\Sigma}) - \lambda_2(\mathbf{\Sigma}) > 0$, fix any $\delta \in (0,1)$, and pick $c := \frac{c_0}{2\mathtt{gap}}$ for any $c_0 > 2$. Next, suppose $R_s \leq BR_e$ (i.e., no discarded data) and define*

$$Q_1 := \frac{64ed\kappa^4\max(1,c^2)}{\delta^2}\ln\frac{4}{\delta}, \quad Q_2 := \frac{512e^2d^2\sigma_B^2\max(1,c^2)}{\delta^4}\ln\frac{4}{\delta}, \tag{22}$$

pick any $Q \geq Q_1 + Q_2$, and choose the stepsize sequence as $\eta_t := c/(Q+t)$. Then, we have for DM-Krasulina that there exists a sequence $(\Omega_t^{'})_{t\in\mathbb{Z}_+}$ of nested subsets of the sample space Ω such that $\mathbb{P}\left(\cap_{t>0}\Omega_t^{'}\right) \geq 1-\delta$ and

$$\mathbb{E}_t\left\{f(\mathbf{w}_t)\right\} - f(\mathbf{w}^*) \leq C_1\Big(\frac{Q+1}{t+Q+1}\Big)^{\frac{c_0}{2}} + C_2\Big(\frac{\sigma_B^2}{t+Q+1}\Big), \tag{23}$$

where $\mathbb{E}_t$ is the conditional expectation over $\Omega_t^{'}$, and C_1 and C_2 are constants defined as

$$C_1 := \frac{\lambda_1(\mathbf{\Sigma})}{2}\left(\frac{4ed}{\delta^2}\right)^{\frac{5}{2\ln 2}} e^{2c^2\lambda_1^2(\mathbf{\Sigma})/Q} \quad \text{and} \quad C_2 := \frac{2c^2\lambda_1(\mathbf{\Sigma})e^{(c_0+2c^2\lambda_1^2(\mathbf{\Sigma}))/Q}}{(c_0-2)}.$$

Theorem 5 is similar in flavor to Theorem 4 in the sense that their respective excess risk bounds in (19) and (23) have (asymptotically) dominant error terms that involve the noise variance (gradient noise for the convex problem and covariance noise for the 1-PCA problem). In particular, since $\sigma_B^2 \leq \sigma^2/B$, the excess risk $f(\mathbf{w}_t) - f(\mathbf{w}^*)$ in DM-Krasulina can be driven down faster by increasing the mini-batch size B up to a certain limit, as noted in the next result. But the two theorems also have some key differences, which can be attributed to DM-Krasulina's focus on global convergence for the nonconvex 1-PCA problem. The first difference is that the excess risk in (23) is being bounded in expectation over a subset of the sample space, whereas the expectation in Theorem 4 is over the whole sample space Ω. The second difference is that the result in Theorem 4 is independent of the ambient dimension d, whereas the result for the 1-PCA problem has d^4 dependence.

We now provide a corollary of Theorem 5 that highlights the speed-up gains associated with DM-Krasulina as long as the mini-batch size B does not exceed a certain limit.

Corollary 1. *Let the parameters and constants be as specified in Theorem 5. Next, pick parameters (Q_1', Q_2') such that $Q_1' \geq Q_1$ and $Q_2' \geq Q_2/\sigma_B^2$, and denote the total number of samples processed by DM-Krasulina as $t' := tB$.*

Then, as long as assumptions from Theorem 5 hold and the network-wide mini-batch size satisfies $B \leq (t')^{1-\frac{2}{c_0}}$, *there exists a sequence* $(\Omega_t^{'})_{t\in\mathbb{Z}_+}$ *of nested subsets of the sample space* Ω *such that* $\mathbb{P}\left(\cap_{t>0}\Omega_t^{'}\right) \geq 1-\delta$ *and*

$$\mathbb{E}_t\left\{f(\mathbf{w}_t)\right\} - f(\mathbf{w}^*) \leq c_0 C_1 \frac{{Q_1'}^{c_0/2}}{t'} + c_0 C_1 \left(\frac{\sigma^2 Q_2'}{t'}\right)^{c_0/2} + \frac{C_2\sigma^2}{t'}. \tag{24}$$

Proof. Substituting $t = t'/B$ in (23) and using simple upper bounds yield

$$\mathbb{E}_t\left\{f(\mathbf{w}_t) - f(\mathbf{w}^*)\right\} \leq C_1\Big(\frac{Q+1}{Q+t}\Big)^{\frac{c_0}{2}} + C_2\Big(\frac{\sigma_B^2}{t}\Big) \leq 2C_1\Big(\frac{Q}{t}\Big)^{\frac{c_0}{2}} + C_2\Big(\frac{\sigma_B^2}{t}\Big).$$

Next, substituting $Q = Q_1' + \sigma_B^2 Q_2'$ in this expression gives us

$$\mathbb{E}_t\left\{\Psi_t\right\} \leq c_0 C_1\Big(\frac{Q_1'}{t}\Big)^{\frac{c_0}{2}} + c_0 C_1\Big(\frac{\sigma_B^2 Q_2'}{t}\Big)^{\frac{c_0}{2}} + C_2\Big(\frac{\sigma_B^2}{t}\Big). \tag{25}$$

Since $\sigma_B^2 \leq \sigma^2/B$ and $t = t'/B$, (25) reduces to the following expression:

$$\mathbb{E}_t\left\{f(\mathbf{w}_t) - f(\mathbf{w}^*)\right\} \leq c_0 C_1 \left(\frac{BQ_1'}{t'}\right)^{c_0/2} + c_0 C_1 \left(\frac{\sigma^2 Q_2'}{t'}\right)^{c_0/2} + \frac{C_2\sigma^2}{t'}.$$

The proof now follows from the assumption that $B \leq (t')^{1-\frac{2}{c_0}}$. ■

In words, Corollary 1 states that DM-Krasulina achieves the optimal excess risk of $O(1/t')$ for the 1-PCA problem, which corresponds to a speed-up gain by a factor of B, as long as $B = O((t')^{1-\frac{2}{c_0}})$ and network resources are provisioned to ensure $R_s \leq BR_e$.

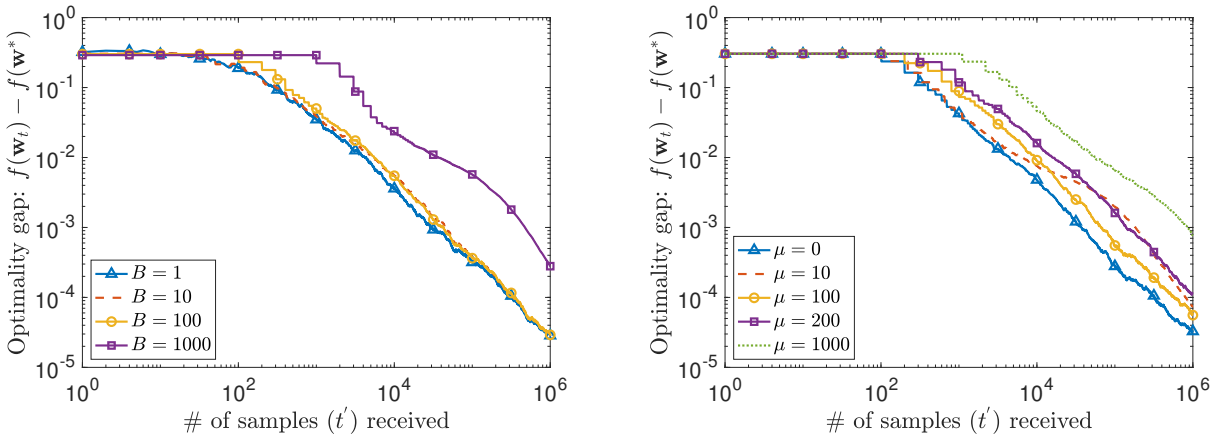
We can also leverage this result to demonstrate how the distributed mini-batch framework of this paper helps us tradeoff computation resources for communication resources. Suppose we are in a compute-rich distributed environment, in which exact averaging requires R rounds of communications, and it is desired to achieve order-optimal risk of $O(1/t')$ for DM-Krasulina. This requires that R_c be fast enough to ensure completion of the communications phase within the time between the end of the computation phase and the arrival of next mini-batch of data; using the definitions from Section II, this means:

$$\frac{R}{R_c} \leq \frac{B}{R_s} - \frac{B}{NR_p} \implies R_c \geq \frac{NRR_sR_p}{B\left(NR_p - R_s\right)}. \tag{26}$$

We can see from this lower bound that increasing the mini-batch size B up to a certain point, while keeping everything else fixed, relaxes the requirement on the communications rate within the network without affecting the quality of the final solution.

We conclude this section by extending Theorem 5 to the under-provisioned setting in which $R_s > BR_e$, possibly due to slower communications links. Similar to our discussion for the DMB algorithm, we express R_s as $R_s = (B+\mu)R_e$ for some $\mu \in \mathbb{Z}_+$ that corresponds to the number of samples that must be discarded at the splitter per iteration due to the mismatch between R_s and BR_e. The following result captures the impact of this data loss on the convergence behavior of DM-Krasulina.

Corollary 2. *Let the parameters and constants be as specified in Corollary 2, and define the final number of algorithmic iterations for DM-Krasulina as* $t^{\mu} := t'/(B+\mu)$. *Then, as long as the assumptions in Theorem 5 hold*



(a) Impact of the mini-batch size on the convergence rate of DM-Krasulina for the resourceful regime. Note that the $B = 1$ plot is effectively Krasulina's method.

(b) Performance of DM-Krasulina in a resource-constrained regime (i.e., $R_s > BR_e$), which causes loss of μ samples per iteration; here, $(N, B) = (10, 100)$.

Fig. 7. Convergence behavior of DM-Krasulina for the case of synthetic data under two scenarios: (a) No data loss ($\mu = 0$) and (b) loss of $\mu > 0$ samples per algorithmic iteration.

and the network-wide mini-batch size satisfies $B \leq (t')^{1-\frac{2}{c_0}}$*, there exists a sequence* $(\Omega_t')_{t\in\mathbb{Z}_+}$ *of nested subsets of the sample space* Ω *such that* $\mathbb{P}\left(\cap_{t>0}\Omega_t'\right) \geq 1-\delta$ *and*

$$\mathbb{E}_{t^\mu}\left\{f(\mathbf{w}_{t^\mu})\right\} - f(\mathbf{w}^*) \leq c_0 C_1 \left(\frac{(B+\mu)Q_1'}{t'}\right)^{c_0/2} + c_0 C_1 \left(\frac{(B+\mu)\sigma^2 Q_2'}{Bt'}\right)^{c_0/2} + \frac{C_2\sigma^2(B+\mu)}{Bt'}. \tag{27}$$

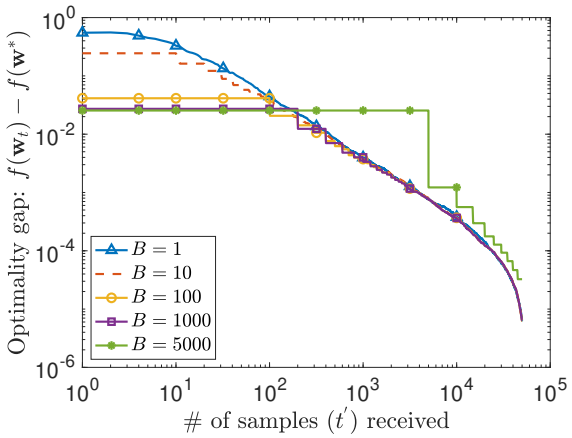
It can be seen from this result that as long as the number of discarded samples per iteration in DM-Krasulina satisfies $\mu = O(B)$, we will have sample-wise order-optimal convergence rate of $O(1/t')$ in the network. This result concerning the impact of discarded samples in under-provisioned distributed systems is similar to the one reported in Theorem 4 for the DMB algorithm.

D. Numerical Experiments for DM-Krasulina

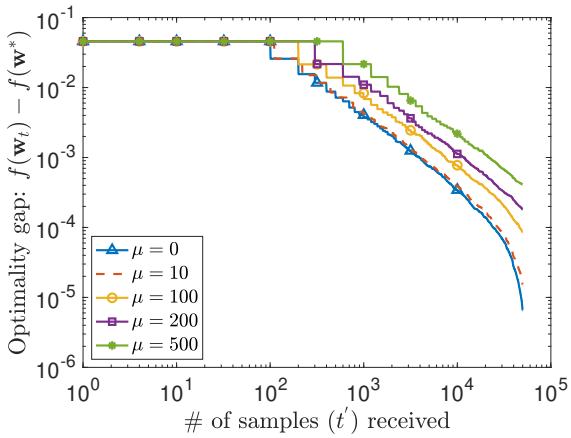
In this section, we provide results of numerical experiments on both synthetic and real-world data to demonstrate the impact of mini-batch size on the performance of DM-Krasulina's method.

1) Synthetic data: For a covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{10\times 10}$ with $\lambda_1(\mathbf{\Sigma}) = 1$ and eigengap $\lambda_1(\mathbf{\Sigma}) - \lambda_2(\mathbf{\Sigma}) = 0.1$, we generate $t' = 10^6$ samples from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. The first experiment in this case deals with the resourceful regime, i.e., $R_S \leq BR_e$, with mini-batches of sizes $B \in \{1, 10, 100, 1000\}$. Results of these experiments are shown in Fig. 7(a), which correspond to stepsize $\eta_t = c/t$ and parameter $c = 10$ that was selected after multiple trial-and-error runs. As predicted by Corollary 1, we see the excess risk after $t = t'/B$ iterations of DM-Krasulina is on the order of $O(1/t')$ for $B \in \{1, 10, 100\}$, while it is not optimal anymore for $B = 1000$.

Next, we demonstrate the performance of DM-Krasulina for resource constrained settings, i.e., $R_s > BR_e$, which causes the system to discard $\mu := (R_s/R_e - B)$ samples per iteration. Using the same data generation setup as before, we run DM-Krasulina for a network of 10 nodes ($N = 10$) with network-wide mini-batch of size $B = 100$



(a) CIFAR-10 Data ($\mu = 0$): Impact of network-wide mini-batch size B on the convergence behavior of DM-Krasulina for the resourceful regime.



(b) CIFAR-10 Data ($N = 10$; $B = 100$): Convergence behavior of DM-Krasulina in a resource-constrained regime, which causes loss of μ samples per iteration.

Fig. 8. Performance of DM-Krasulina for the CIFAR-10 dataset under two scenarios: (a) No data loss ($\mu = 0$) and (b) loss of $\mu > 0$ samples per algorithmic iteration.

(i.e., $B/N = 10$). We consider different mismatch factors between streaming, processing, and communication rates in this experiment, which result in the number of samples being discarded as $\mu \in \{0, 10, 100, 200, 1000\}$. The results are plotted in Fig. 7(b), which show that the values of excess risk for $\mu \in \{10, 100, 200\}$ are comparable to that for $\mu = 0$, but the error for $\mu = 1000$ is an order of magnitude worse than the nominal error.

2) Real-world Data: We next demonstrate the performance of DM-Krasulina on CIFAR-10 dataset [120], which consists of roughly 5×10^4 training samples with $d = 3072$. Our first set of experiments for this dataset uses the stepsize $\eta_t = c/t$ with $c \in \{8 \times 10^4, 8 \times 10^4, 9 \times 10^4, 10^5, 10^5\}$ for network-wide mini-batch sizes $B \in \{1, 10, 100, 1000, 5000\}$ in the resourceful regime ($\mu = 0$). The results, which are averaged over 50 random initializations and random shuffling of data, are given in Fig. 8(a). It can be seen from this figure that the final error relatively stays the same as B increases from 1 to 1000, but it starts getting affected significantly as the network-wide mini-batch size is further increased to $B = 5000$. Our second set of experiments for the CIFAR-10 dataset corresponds to the resource-constrained regime with $(N, B) = (10, 100)$ and stepsize parameter $c = 8 \times 10^4$ for the number of discarded samples $\mu \in \{0, 10, 100, 200, 500\}$. The results, averaged over 200 trials and given in Fig. 8(b), show that the system can tolerate loss of some data samples per iteration without significant increase in the final error; the increase in error, however, becomes noticeable as μ approaches B. Both these observations are in line with the insights of the theoretical results.

V. Distributed Stochastic Approximation Using Inexact Averaging

In this section, we discuss recent results for distributed learning from fast streaming data when the averages computed among nodes in the network are *inexact*. As mentioned earlier, inexact averaging occurs in networks where the communications topology either changes with time or it is unknown in advance to facilitate construction of an

MPI infrastructure for `AllReduce`-style computations. Similar to Section IV, we require here that nodes compute distributed averages of their stochastic gradients in order to reduce their variance and speed up convergence. Unlike Algorithms 1 and 2, however, these averages are computed via R rounds of averaging consensus, as described in Section III-B. This introduces a fundamental trade-off: the more consensus rounds R used per algorithmic iteration, the smaller the effective gradient noise and averaging error, but the longer it takes to complete each iteration.

Similar to the DMB algorithm described in Section IV-A, we mitigate this trade-off by careful mini-batching of gradient samples. Each node n first locally averages the gradients for its local mini-batch of B/N data samples, after which nodes *approximately* average the mini-batched gradients via consensus iterations. Such mini-batching again speeds up the effective processing rate, allowing the network to process more samples. In this section we detail the precise conditions under which this speed-up is enough to achieve near-optimum convergence rates.

We focus exclusively on convex loss functions $\ell(\mathbf{w}, \mathbf{z})$ in this section and present two algorithms for tackling fast streaming data under inexact averaging: *distributed stochastic gradient descent* (D-SGD) and *accelerated distributed stochastic gradient descent* (AD-SGD). Both these algorithms are distributed variants of the SGD and accelerated SGD methods described in Section III-A1: after computing local mini-batch gradients and performing distributed averaging consensus, nodes in these algorithms take (accelerated) SGD steps with respect to the averaged gradients.

A. Algorithms for Distributed Stochastic Convex Approximation

We now present algorithmic details for D-SGD and AD-SGD, both of which are distilled and unified versions of algorithms published in [109], [110].[3] The operating assumption here is that there is sufficient provisioning of resources within the system to ensure $R_s \leq BR_e$. In the next section, we present the convergence rates of D-SGD and AD-SGD under this assumption, and explicitly describe the regimes of system parameters in which these methods have optimum convergence rates. An important feature of these forthcoming results is the impact of acceleration in these regimes. Accelerated methods provide additional "headroom," allowing for larger mini-batches that can process faster data streams while still yielding optimum convergence rates.

The algorithmic descriptions in the following make use of a symmetric, doubly stochastic matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ that is consistent with the network graph G, as discussed in Section III-B2. We suppose that the second-largest eigenvalue magnitude obeys $|\lambda_2(\mathbf{A})| < 1$, where the inequality must be strict. This rather mild assumption is guaranteed, *inter alia*, by choosing $\mathbf{A}$ to have elements strictly greater than zero for all elements corresponding to an edge of a connected graph G, i.e., each node includes its entire neighborhood (including itself) in the local convex combination it computes for each consensus round.

1) Description of D-SGD: The D-SGD algorithm generalizes SGD with Polyak–Ruppert averaging to the setting of distributed, streaming data. We mathematically detail the steps of D-SGD in Algorithm 3, and summarize them here. At the beginning of every iteration t of D-SGD, each node n receives a mini-batch $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ of B/N i.i.d. data samples. Each node n afterwards computes $\mathbf{g}_{n,t}$, the average gradient over its local mini-batch, and then

[3]In particular, [109] presents a distributed learning strategy based on *dual averaging*, a method for stochastic convex optimization that has convergence rates similar to those of SGD. In contrast, [110] presents a strategy based on *mirror descent*, a generalization of SGD-style methods. For clarity of exposition, we present results in here under the SGD framework.

Algorithm 3 Distributed Stochastic Gradient Descent (D-SGD)

Require: Provisioning of compute and communications resources to ensure $R_s \leq BR_e$

Input: Data stream $\{\mathbf{z}_{t'} \overset{\text{i.i.d.}}{\sim} \mathcal{D}\}_{t' \in \mathbb{Z}_+}$ that is split into N streams of mini-batched data $\{\mathbf{z}_{n,b,t}\}_{b=1,t\in\mathbb{Z}_+}^{B/N}$ across N nodes, doubly stochastic matrix $\mathbf{A}$, number of consensus rounds R, and stepsize sequence $\{\eta_t \in \mathbb{R}_+\}_{t\in\mathbb{Z}_+}$

Initialize: All compute nodes initialize with $\mathbf{w}_{n,0} = \mathbf{0} \in \mathbb{R}^d$

for $t = 1, 2, \ldots,$ **do**
  $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{0} \in \mathbb{R}^d$
  for $b = 1, \ldots, B/N$ **do** ▷ Node n receives the mini-batch $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ and updates $\mathbf{g}_{n,t}$ locally
    $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,b,t} \leftarrow \nabla \ell(\mathbf{w}_{n,t}, \mathbf{z}_{n,b,t})$
    $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{g}_{n,t} + \frac{1}{B/N}\mathbf{g}_{n,b,t}$
  end for
  $\forall n \in \{1, \ldots, N\},\ \mathbf{h}_{n,t,0} \leftarrow \mathbf{g}_{n,t}$ ▷ Get mini-batched gradients
  for $r = 1, \ldots, R$ and $n = 1, \ldots, N$ **do** ▷ R rounds of averaging consensus
    $\mathbf{h}_{n,t,r} \leftarrow \sum_{m=1}^{N} a_{n,m}\mathbf{h}_{m,t,r-1}$
  end for
  for $n = 1, \ldots, N$ **do**
    $\mathbf{w}_{n,t+1} \leftarrow [\mathbf{w}_{n,t} - \eta_t \mathbf{h}_{n,t,R}]_{\mathcal{W}}$ ▷ Projected SGD step
    $\mathbf{w}_{n,t+1}^{\mathsf{av}} \leftarrow \left(\sum_{\tau=0}^{t} \eta_\tau\right)^{-1} \sum_{\tau=0}^{t} \eta_\tau \mathbf{w}_{n,\tau+1}$ ▷ Averaging of iterates
  end for
end for

Return: Decentralized estimates $\{\mathbf{w}_{n,t}^{\mathsf{av}}\}_{n\in\mathcal{V}}$ of the Bayes optimal solution after receiving $t' = Bt$ samples

engages in $R \in \mathbb{Z}_+$ rounds of averaging consensus, where the parameter R satisfies the constraints in (3). This results in an approximate average of all N mini-batches at each node n, which we denote by $\mathbf{h}_{n,t,R}$. Each node finally takes an SGD step using $\mathbf{h}_{n,t,R}$ and engages in Polyak–Ruppert averaging to obtain the estimate $\mathbf{w}_{n,t}^{\mathsf{av}}$.

2) Description of AD-SGD: The AD-SGD algorithm generalizes *accelerated* SGD, as presented in Section III-A1, to the distributed setting. The generalization is similar to D-SGD's extension of the SGD procedure: Nodes collect mini-batches $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ from their data streams, compute average gradients $\mathbf{g}_{n,t}$, and get approximate gradient averages $\mathbf{h}_{n,t,R}$ via R rounds of averaging consensus. However, instead of taking a standard SGD step, compute nodes take an accelerated SGD step using $\mathbf{h}_{n,t,R}$. This involves each node maintaining iterates $\mathbf{u}_{n,t}$, $\mathbf{v}_{n,t}$, and $\mathbf{w}_{n,t}$ as in accelerated SGD, which are averaged and updated according to two sequences of stepsizes $\beta_t \in [1, \infty)$ and $\eta_t \in \mathbb{R}_+$. We mathematically detail the steps of AD-SGD in Algorithm 4.

B. Convergence Results and Scaling Laws

Here, we present results on the convergence speeds of D-SGD and AD-SGD as well as their gap to the ideal case in which data streams can be centrally processed by a single powerful machine, or, equivalently, compute nodes can process instantaneously and communicate at infinite rates. We begin by presenting results for D-SGD.

Algorithm 4 Accelerated Distributed Stochastic Gradient Descent (AD-SGD)

Require: Provisioning of compute and communications resources to ensure $R_s \leq BR_e$

Input: Incoming mini-batched data streams at N compute nodes, expressed as $\{\mathbf{z}_{n,b,t}\}_{b=1,t\in\mathbb{Z}_+}^{B/N}$, doubly stochastic matrix $\mathbf{A}$, number of consensus rounds R, and stepsize sequences $\{\eta_t, \beta_t \in \mathbb{R}_+\}_{t\in\mathbb{Z}_+}$

Initialize: All compute nodes initialize with $\mathbf{u}_{n,0}, \mathbf{v}_{n,0}, \mathbf{w}_{n,0} = \mathbf{0} \in \mathbb{R}^d$

for $t = 1, 2, \ldots,$ **do**
  $\forall n \in \{1, \ldots, N\},\ \mathbf{u}_{n,t} \leftarrow \beta_t^{-1}\mathbf{v}_{n,t} + (1 - \beta_t^{-1})\mathbf{w}_{n,t}$
  $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{0} \in \mathbb{R}^d$
  for $b = 1, \ldots, B/N$ **do** ▷ Node n receives the mini-batch $\{\mathbf{z}_{n,b,t}\}_{b=1}^{B/N}$ and updates $\mathbf{g}_{n,t}$ locally
    $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,b,t} \leftarrow \nabla\ell(\mathbf{u}_{n,t}, \mathbf{z}_{n,b,t})$
    $\forall n \in \{1, \ldots, N\},\ \mathbf{g}_{n,t} \leftarrow \mathbf{g}_{n,t} + \frac{1}{B/N}\mathbf{g}_{n,b,t}$
  end for
  $\forall n \in \{1, \ldots, N\},\ \mathbf{h}_{n,t,0} \leftarrow \mathbf{g}_{n,t}$ ▷ Get mini-batched gradients
  for $r = 1, \ldots, R$ and $n = 1, \ldots, N$ **do** ▷ R rounds of averaging consensus
    $\mathbf{h}_{n,t,r} \leftarrow \sum_{m=1}^{N} a_{n,m}\mathbf{h}_{m,t,r-1}$
  end for
  for $n = 1, \ldots, N$ **do** ▷ Project A-SGD step
    $\mathbf{v}_{n,t+1} \leftarrow [\mathbf{u}_{n,t} - \eta_t\mathbf{h}_{n,t,r}]_{\mathcal{W}}$
    $\mathbf{w}_{n,t+1} \leftarrow \beta_t^{-1}\mathbf{v}_{n,t+1} + (1 - \beta_t^{-1})\mathbf{w}_{n,t}$
  end for
end for

Return: Decentralized estimates $\{\mathbf{w}_{n,t}\}_{n\in\mathcal{V}}$ of the Bayes optimal solution after receiving $t' = Bt$ samples

Theorem 6. *Let the loss function $\ell(\mathbf{w}, \mathbf{z})$ be convex and smooth with L-Lipschitz gradients and gradient noise variance σ^2, and suppose a bounded model space with expanse $D_{\mathcal{W}}$. Further suppose that nodes engage in R rounds of consensus averaging in each iteration of D-SGD. Then, there exist stepsizes η_t such that the expected excess risk at each node n after t iterations is bounded by*

$$\mathbb{E}\big\{f(\mathbf{w}_n^{\mathsf{av}}(t))\big\} - f(\mathbf{w}^*) \leq \frac{2L}{t} + \sqrt{\frac{4\Delta_t^2}{t}} + \sqrt{\frac{1}{2}}\frac{\Xi_t D_{\mathcal{W}}}{L}, \tag{28}$$

where

$$\Xi_t := \left(\frac{\sigma}{\sqrt{B/N}}\right)(1 + N^2|\lambda_2(\mathbf{A})|^R)((1 + N^2|\lambda_2(\mathbf{A})|^R)^t - 1) \tag{29}$$

and

$$\Delta_t^2 := 4\sigma^2/B + 2\left(\frac{\sigma}{\sqrt{B/N}}\right)^2(1 + N^4|\lambda_2(\mathbf{A})|^{2R})((1 + N^2|\lambda_2(\mathbf{A})|^R)^t - 1)^2 + 4|\lambda_2(\mathbf{A})|^{2R}\sigma^2N^3/B \tag{30}$$

quantify the moments of the effective gradient noise.

Proof Sketch. The convergence analysis follows that of standard results of SGD-style methods, with careful analysis of the equivalent gradient noise resulting from distributed consensus averaging. In particular, Δ_t^2 bounds the equivalent noise variance, and it has two components: $4\sigma^2/B$, which is the equivalent variance if distributed gradients are averaged exactly, and additional terms that express variance increases due to inexact averaging. These latter terms go to zero geometrically as $R \to \infty$. The term Ξ_t bounds the gradient *bias*, which is entirely due to averaging errors and which also goes to zero as $R \to \infty$. Inserting these bounds on the bias and variance into the analysis of SGD (see, e.g., [3]) gives the result. See [110] for a detailed proof. ■

Remark 5. The result in Theorem 6 is stated in terms of the ideal stepsizes discussed in Remark 2, which suppose a finite and known time horizon t. Although the time horizon may not be known in advance, the conditions for optimality discussed in the following depend on the time horizon; as such, we retain these ideal stepsizes to make explicit this dependence of the results on the time horizon / final iteration count t.

The convergence rate in Theorem 6 has the same form as that of standard SGD given in Theorem 1. The main difference is in the bias and variance of the stochastic gradients, which depend on the processing and communications rates relative to the data streaming rate, and therefore on how many rounds of averaging consensus nodes can carry out per iteration of D-SGD.

The critical question for D-SGD is how fast communication needs to be for order-optimum convergence speed, i.e., the convergence speed that one would obtain if nodes had noiseless access to other nodes' gradient estimates in each iteration. Recall that the system has received $t' = tB$ data samples after t data-splitting instances. Centralized SGD—with access to all tB data samples in sequence—achieves the convergence rate $O\left(\frac{L}{Bt} + \frac{\sigma}{\sqrt{Bt}}\right)$, where the final term dominates the error as a function of t if $\sigma^2 > 0$. In the following corollary we state conditions under which the convergence rate of D-SGD matches this term.

Corollary 3. *The optimality gap for D-SGD at each node n satisfies*

$$\mathbb{E}\big\{f(\mathbf{w}_n^{\mathsf{av}}(t))\big\} - f(\mathbf{w}^*) = O\left(\frac{\sigma}{\sqrt{t'}}\right), \tag{31}$$

provided the network-wide mini-batch size B, the communications rate R_c, and the number of nodes N satisfy

$$B/N = \Omega\left(1 + \frac{\log(t')}{\rho \log(1/|\lambda_2(\mathbf{A})|)}\right), \quad B/N = O\left(\frac{\sigma\sqrt{t'}}{N}\right),$$
$$R_c = \Omega\left(\frac{R_s \log(t')}{\sigma\sqrt{t'}\log(1/|\lambda_2(\mathbf{A})|)} + \frac{R_s}{R_p N}\right), \quad t' = \Omega\left(\frac{N^2}{\sigma^2}\right),$$

where

$$\rho := N\frac{R_c}{R_s} - \frac{1}{R_p}$$

is the ratio of the "effective" communications rate per sample, which discounts the time spent in computation, and the rate at which streaming data arrive at the system.

This corollary describes the dependence of the convergence rate of D-SGD on the processing, communications and streaming rates, network topology, and network-wide mini-batch size. We now point out a few connections

between this result and the one for the DMB algorithm described in Section IV-A. In the case of the DMB algorithm, recall that a maximum local mini-batch size of $B/N = O(\sqrt{t'}/N)$ is prescribed to ensure that the $O(1/t')$ term in the SGD convergence bound does not dominate. Corollary 3 further prescribes a *minimum* local mini-batch size B/N needed to ensure that nodes have time to carry out sufficient consensus. This condition fails to obtain when the communications rate is too slow to accommodate the required mini-batch size. Indeed, for all else constant, the optimum local mini-batch size is merely $\Omega(\log(t'))$, and the condition on R_c essentially ensures $B/N = O(t^{1/2})$.

Further, Corollary 3 dictates the relationship between the size of the network and the total number of data samples obtained at each node. Leaving the other terms constant, Corollary 3 requires $t' = \Omega(N^2)$, which implies that the total number of data samples processed *per node* should scale faster than the number of nodes in the network. This is a relatively mild condition for big data applications; indeed, many applications involve data streams that are large relative to the size of the network.

Along similar lines, ignoring other constants and log terms, Corollary 3 indicates that a communications rate of $R_c = \Omega\big(R_s/\sqrt{t'} + R_s/(R_p N)\big)$ is sufficient for order optimality. Thus, if the total number of data samples *per node* grows faster than the number of nodes, the required communications rate approaches the ratio between the sampling and processing rates of the network, i.e., *communications need only be fast enough that processing is not the bottleneck*. This implies that for fixed networks or network families in which the spectral gap $1 - |\lambda_2(\mathbf{A})|$ is bounded away from zero, even slow communications is sufficient for near-optimum learning.

Next, we bound the expected gap to optimality of the AD-SGD iterates.

Theorem 7. *Let the loss function $\ell(\mathbf{w}, \mathbf{z})$ be convex and smooth with L-Lipschitz gradients and gradient noise variance σ^2, and suppose a bounded model space with expanse $D_{\mathcal{W}}$. Further suppose nodes engage in R rounds of consensus averaging in each iteration of AD-SGD, and use stepsizes $\eta_t = (t+1)/2\eta$, for $0 < \eta < 1/(2L)$ and $\beta_t = (t+1)/2$. Then, the expected excess risk at each node n after t iterations of AD-SGD is bounded by*

$$\mathbb{E}\{f(\mathbf{w}_n(t))\} - f(\mathbf{w}^*) \leq \frac{8L}{t^2} + 4\sqrt{\frac{4\Delta_t^2}{t}} + \sqrt{32}\Xi_t,$$

where

$$\Delta_t^2 = 2(\sigma/\sqrt{B/N})^2((1 + 2\eta_t N^2 L|\lambda_2(\mathbf{A})|^R)^t - 1)^2 + \frac{4\sigma^2}{B/N}(|\lambda_2(\mathbf{A})|^{2R} N^2 + 1/N)$$

and

$$\Xi_t = (\sigma/\sqrt{B/N})(1 + B^2|\lambda_2(\mathbf{A})|^R)((1 + 2\eta_t N^2 L|\lambda_2(\mathbf{A})|^R)^t - 1).$$

Proof sketch. The result again follows from a careful analysis of the bias and variance of the equivalent gradient noise. As before, as $R \to \infty$, the variance term Δ_t^2 has order $O(\sigma^2/B)$ and the bias term Ξ_t vanishes. Putting these quantities into the analysis of accelerated SGD gives the result; we refer the reader to [110] for details. ■

Remark 6. Theorem 7 is also stated in terms of the ideal stepsizes given in [3], which suppose a finite and known time horizon; we again retain this form of the result in order to keep explicit the dependence on time horizon.

Corollary 4. *The excess risk for AD-SGD at each node n satisfies*

$$\mathbb{E}\{f(\mathbf{w}_n(t)\} - f(\mathbf{w}^*) = O\left(\frac{\sigma}{\sqrt{t'}}\right),$$

provided

$$B/N = \Omega\left(1 + \frac{\log(t')}{\rho \log(1/|\lambda_2(\mathbf{A})|)}\right), \quad B/N = O\left(\frac{\sigma^{1/2}(t')^{3/4}}{N}\right),$$
$$R_c = \Omega\left(\frac{R_s \log(t')}{\sigma (t')^{3/4} \log(1/|\lambda_2(\mathbf{A})|)} + \frac{R_s}{R_p N}\right), \quad t' = \Omega\left(\frac{N^{4/3}}{\sigma^2}\right),$$

where again $\rho := N\frac{R_c}{R_s} - \frac{1}{R_p}$ is the ratio of the effective communications rate per sample and the streaming rate.

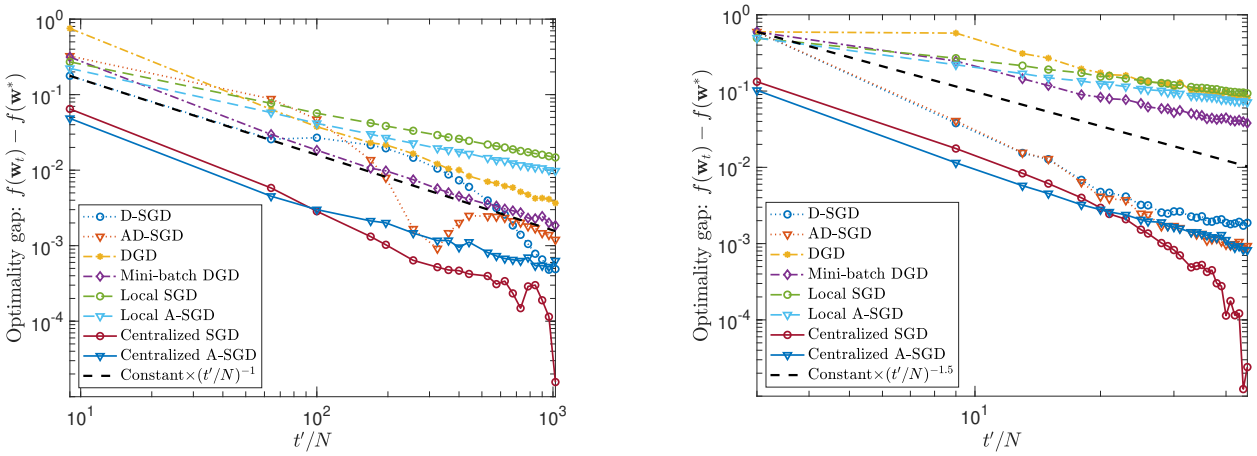
Notice that the crucial difference between the two schemes is that AD-SGD has a convergence rate of $1/t^2$ in the absence of noise. This faster term, which is often negligible in centralized SGD, means that AD-SGD tolerates more aggressive mini-batching without an impact on the order of the convergence rate. As a result, the condition on R_c is relaxed by $1/4$ in the exponent of t'. This is because the condition $B/N = O(t^{1/2})$, which holds for standard stochastic methods, is relaxed to $B/N = O(t^{3/4})$ for accelerated SGD. Thus, the use of accelerated methods increases the domain in which order-optimum rate-limited learning is guaranteed.

C. Numerical Experiments for D-SGD and AD-SGD

In order to demonstrate the scaling laws predicted by Corollaries 3 and 4 and to investigate the empirical performance of D-SGD and AD-SGD, we once again resort to binary linear classification using logistic regression. Similar to Section IV-B, we examine performance on synthetic data in order to have a "ground truth" data distribution against which to compare performance. For the sake of richness, we generate data in here using a slightly different probabilistic model than in Section IV-B. Specifically, we suppose the data follow conditional Gaussian distributions: For $y_{t'} \in \{-1, +1\}$, we let $\mathbf{x}_{t'} \sim \mathcal{N}(\mu_{y_{t'}}, \sigma_x^2 \mathbf{I})$, where $\mu_{y_{t'}} \in \{\mu_{-1}, \mu_1\}$ is one of two mean vectors, and $\sigma_x^2 > 0$ is the noise variance (not to be confused with gradient noise variance σ^2.) For the experiments, we pick $d = 20$, choose $\sigma_x^2 = 2$, and draw the elements μ_{-1} and μ_1 randomly from the standard normal distribution.

We compare the performance of D-SGD and AD-SGD against several other schemes. As a best-case scenario, we consider *centralized* counterparts of D-SGD and AD-SGD, meaning that all B data samples and their associated gradients at each data-splitting instance are available at a single machine, which carries out SGD and *accelerated* SGD. Both these algorithms naturally have the best average performance. As a baseline, we consider *local* (accelerated) SGD, in which nodes simply perform SGD on their own data streams without collaboration. This scheme benefits from an insensitivity to the *mismatch ratio* ρ (defined in Corollary 4), i.e., it does not require any mini-batching, and therefore it represents a minimum standard for performance.

Finally, we consider a communications-constrained adaptation of DGD [121] (see Section III-B for a description of DGD). Note that DGD implicitly supposes $\rho = 1$; to handle the $\rho < 1$ case, we consider two adaptations: *naive* DGD, in which data samples that arrive between computation+communications rounds are simply discarded, and *mini-batched* DGD, in which nodes compute *local* mini-batches of size $B/N = 1/\rho$, take gradient updates using the local mini-batch, and carry out a consensus round. While it is not designed for the communications-limited



(a) $t' = N^2$ (b) $t' = N^{3/2}$

Fig. 9. Performance of different distributed and centralized first-order methods on 6-regular expander graphs, for $\rho = 1/2$, measured in terms of the excess risk for binary logistic regression.

scenario, DGD has good performance in general, so it represents a natural alternative against which to compare the performance of D-SGD and AD-SGD.

For network topology, we use *expander graphs*, which are families of graphs that have spectral gap $1 - |\lambda_2(\mathbf{A})|$ bounded away from zero as $N \to \infty$. In particular, we use 6-regular graphs, i.e., regular graphs in which each node has six neighbors, drawn uniformly from the ensemble of such graphs. Because $|\lambda_2(\mathbf{A})|$ is strictly bounded above by 1 for expander graphs, one can more easily examine whether performance of D-SGD and AD-SGD agrees with the ideal scaling laws discussed in Corollaries 3 and 4. At the same time, because D-SGD and AD-SGD make use of imperfect averaging, expander graphs also allow us to examine non-asymptotic behavior of the two schemes. Per Corollaries 3 and 4, we choose $B/N = \frac{1}{10}\frac{\log(t')}{\rho \log(1/|\lambda_2(\mathbf{A})|)}$. While such scaling is guaranteed to be sufficient for optimum asymptotic performance, we chose the multiplicative constant $1/10$ via trial-and-error to give good non-asymptotic performance.

In Fig. 9 we plot the performance of different methods averaged over 600 Monte Carlo trials. We take $\rho = 1/2$, and consider the regimes $t' = N^2$ (Fig. 9(a)) and $t' = N^{3/2}$ (Fig. 9(b)). For D-SGD, the stepsizes are taken to be $\eta_t = 2.5/\sqrt{t}$. For AD-SGD, we take $\beta_t = t/2$ as prescribed in [3], as well as $\eta_t = 8/(t+1)^{3/2}$ when $t' = N^{3/2}$ and $\eta_t = 14/(t+1)^{3/2}$ when $t' = N^2$. We arrived at the constants in front of η_t via trial-and-error. We see that AD-SGD and D-SGD outperform local methods, while their performance is roughly in line with asymptotic theoretical predictions. The performance of DGD, on the other hand, depends on the regime: For $t' = N^2$, it appears to have order-optimum performance, whereas for $t' = N^{3/2}$ it has suboptimum performance on par with local methods. The reason for the dependency of DGD on regime is not immediately clear and suggests the need for further study into DGD-style methods in the case of rate-limited networks.

VI. Conclusion and Future Directions

The development, characterization, and implementation of efficient learning from fast (and distributed) streams of data is a challenge for researchers and practitioners for the coming decade. In this paper, we have laid out results that suggest that such learning is possible—even when the processing rates of individual compute nodes and/or network communications are slow relative to the streaming rate of data. In particular, we have framed this problem as a distributed stochastic approximation problem, in which streams of independent and identically distributed (i.i.d.) data samples arrive at compute nodes that are connected by communications networks and that exchange messages at a fixed rate—which may be slower than the rate at which samples arrive.

Within this framework, we have discussed distributed first-order stochastic optimization methods that efficiently solve the learning problem, both in systems with robust communications networks that can implement exact `AllReduce`-style aggregation of data, and in systems with decentralized communications networks that implement approximate aggregation of data via averaging consensus. We have also presented performance guarantees for these methods for general convex problems and for the "well-behaved" nonconvex problem of principal component analysis (PCA), for both of which global optimization is possible.

A critical component of these methods is explicit *local mini-batching*, in which nodes average together the gradients (or gradient-like quantities) of multiple samples. Nodes then need only communicate the gradient of the entire local mini-batch, which substantially reduces the communications burden of distributed learning. A consistent through-line of these results is that both the (implicit) network-wide and the (explicit) local mini-batch sizes must be chosen carefully; small mini-batches do not slow down algorithmic iterations sufficiently to counter the fast streaming rate and/or reduce the communications load, and large mini-batches slow them down so far that convergence is slowed down. We give both a precise characterization of the necessary mini-batch size and the network constraints—in terms of the size, topology, and communications rates—under which it is possible to obtain convergence rates that are as fast (order-wise) as the ideal case in which there are ample compute nodes and communications between the nodes is perfect and instantaneous. Perhaps surprisingly, these results show that even relatively slow communication links are often sufficient for optimal distributed learning.

We conclude this paper with discussion of a subset of research questions that these results leave open.

Nonconvex losses. The results presented here are for convex losses or for the PCA problem, a special case in which all local optima are global, and first-order methods can converge on a global optimum. But many important machine learning problems, including deep learning, are highly nonconvex. Recently, several works have proposed and analyzed methods for distributed nonconvex optimization [101]–[103]. However, to the best of our knowledge, the question of general nonconvex learning from fast, distributed data streams with global guarantees remains open.

One of the major challenges in extending the results presented here is pinpointing the impact of inexact averaging on the potential mini-batching gains. When nodes compute inexact averages of their gradients, their iterates may diverge. This divergence may be catastrophic in the case of nonconvex learning, as nodes' iterates may end up in different basins of attraction. In this scenario, gradients will be taken with respect to increasingly distant operating points, and averaging them according to the recipe of this paper may not result in good search directions.

Message quantization. This work supposes nodes exchange real-valued messages, whereas messages are quantized in digital communications networks. Further, in addition to local mini-batching, nodes can speed up communications by explicitly quantizing their messages, thereby reducing the network throughput required to exchange messages. But quantization introduces additional noise into the system, which requires further analysis and algorithmic control. A variety of methods have been proposed for tackling quantization-aware learning, both in centralized and distributed settings [122]–[124], including *signSGD* [125], in which nodes exchange one-bit quantized gradients. An open question is when and whether such schemes are optimal for distributed learning from fast streaming data.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[2] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Intl. Conf. Computational Statistics (COMPSTAT'2010)*, Paris, France, Aug. 2010, pp. 177–187.

[3] G. Lan, "An optimal method for stochastic composite optimization," *Mathematical Programming*, vol. 133, no. 1-2, pp. 365–397, 2012.

[4] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'13)*, 2013, pp. 315–323.

[5] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. 20th ACM Intl. Conf. Knowledge Discovery and Data Mining (SIGKDD'14)*, 2014, pp. 661–670.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[7] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[8] J. Tsitsiklis and M. Athans, "Convergence and asymptotic agreement in distributed decision problems," *IEEE Trans. Automatic Control*, vol. 29, no. 1, pp. 42–50, Jan. 1984.

[9] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[10] 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, Aug. 2012.

[11] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Automatic Control*, vol. 31, no. 9, pp. 803–812, Sep. 1986.

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

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Operating Systems Design Implementation (OSDI'04)*, San Francisco, CA, 2004, pp. 1–10.

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

[15] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Rev.*, vol. 53, no. 4, pp. 747–772, 2011.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers*, ser. Foundations and Trends in Machine Learning. Hanover, MA: Now Publishers Inc., Jan. 2011, vol. 3, no. 1.

[17] A. H. Sayed, *Adaptation, Learning, and Optimization Over Networks*, ser. Foundations and Trends in Machine Learning. Hanover, MA: Now Publishers Inc., Jul. 2014, vol. 7, no. 4-5.

[18] A. Nedić, "Distributed optimization over networks," in *Multi-agent Optimization*, F. Facchinei and J.-S. Pang, Eds. Springer International Publishing, 2018, pp. 1–84.

[19] A. Nedić, A. Olshevsky, and M. G. Rabbat, "Network topology and communication–computation tradeoffs in decentralized optimization," *Proc. IEEE*, vol. 106, no. 5, pp. 953–976, May 2018.

[20] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, 2020.

[21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascon, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecny, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Ozgur, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramer, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[22] Z. Yang, A. Gang, and W. U. Bajwa, "Adversary-resilient distributed and decentralized statistical inference and machine learning: An overview of recent advances under the Byzantine threat model," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 146–159, 2020.

[23] R. Xin, S. Kar, and U. A. Khan, "Decentralized stochastic optimization and machine learning: A unified variance-reduction framework for robust performance and fast convergence," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 102–113, 2020.

[24] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Operating Systems Design and Implementation (OSDI'14)*, 2014, pp. 583–598.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Systems Design and Implementation (OSDI'16)*, Savannah, GA, Nov. 2016, pp. 265–283.

[26] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker, "An introduction to the MPI standard," University of Tennessee, Tech. Rep., 1995.

[27] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proc. 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, Sep. 2004, pp. 97–104.

[28] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'11)*, 2011, pp. 693–701.

[29] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," in *Proc. 50th Annu. Allerton Conf. Communication, Control, and Computing*, Oct. 2012, pp. 1543–1550.

[30] H. Zhang, C.-J. Hsieh, and V. Akella, "Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent," in *Proc. IEEE 16th Intl. Conf. Data Mining (ICDM'16)*, 2016, pp. 629–638.

[31] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proc. Intl. Conf. Learning Representations (ICLR'16) Workshop Track*, 2016.

[32] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. 34th Intl. Conf. Machine Learning (ICML'17)*, Sydney, Australia, Aug. 2017, pp. 3368–3376.

[33] J. Wang, V. Tantia, N. Ballas, and M. Rabbat, "SlowMo: Improving communication-efficient distributed SGD with slow momentum," in *Proc. Intl. Conf. Learning Representations (ICLR'20)*, 2020.

[34] V. Vapnik, "An overview of statistical learning theory," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 988–999, 1999.

[35] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan, "Stochastic convex optimization," in *Proc. Conf. Learning Theory (COLT'09)*, 2009.

[36] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proc. 28th Intl. Conf. Machine Learning (ICML'11)*, Madison, WI, 2011, pp. 265–272.

[37] B. Neyshabur, R. R. Salakhutdinov, and N. Srebro, "Path-SGD: Path-normalized optimization in deep neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS'15)*, 2015, pp. 2422–2430.

[38] J. C. Spall, "Stochastic optimization," in *Handbook of Computational Statistics: Concepts and Methods*, J. E. Gentle, W. K. Härdle, and Y. Mori, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 173–201.

[39] S. Kim, R. Pasupathy, and S. G. Henderson, "A guide to sample average approximation," in *Handbook of Simulation Optimization*, M. C. Fu, Ed. New York, NY: Springer New York, 2015, pp. 207–243.

[40] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 1951.

[41] E. Moulines and F. R. Bach, "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," in *Proc. Advances in Neural Information Processing Systems (NeurIPS'11)*, 2011, pp. 451–459.

[42] O. Shamir, "Convergence of stochastic gradient descent for PCA," in *Proc. 33rd Intl. Conf. Machine Learning (ICML'16)*, New York, NY, Jun. 2016, pp. 257–265.

[43] A. Bijral, A. Sarwate, and N. Srebro, "Data-dependent convergence for consensus stochastic optimization," *IEEE Trans. Autom. Control*, vol. 62, no. 9, pp. 4483–4498, Sep. 2017.

[44] R. Frostig, R. Ge, S. M. Kakade, and A. Sidford, "Competing with the empirical risk minimizer in a single pass," in *Proc. 28th Conf. Learning Theory (COLT'15)*, Jul. 2015, pp. 728–763.

[45] B. Polyak, "A new method of stochastic approximation type," *Avtomat. i Telemekh*, no. 7, pp. 98–107, 1990.

[46] D. Ruppert, "Efficient estimators from a slowly convergent Robbins-Monro process," School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, Tech. Rep. No. 781, 1988.

[47] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control and Opt.*, vol. 30, no. 4, pp. 838–855, 1992.

[48] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Dokl. Akad. Nauk SSSR*, vol. 269, 1983, pp. 543–547.

[49] H. Kushner and G. G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Science & Business Media, 2003, vol. 35.

[50] A. Agarwal, M. J. Wainwright, P. L. Bartlett, and P. K. Ravikumar, "Information-theoretic lower bounds on the oracle complexity of convex optimization," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'09)*, 2009, pp. 1–9.

[51] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *J. Machine Learning Res.*, vol. 11, pp. 2543–2596, Oct. 2010.

[52] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM J. Opt.*, vol. 23, no. 4, pp. 2341–2368, 2013.

[53] ——, "Accelerated gradient methods for nonconvex nonlinear and stochastic programming," *Mathematical Programming*, vol. 156, no. 1-2, pp. 59–99, 2016.

[54] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points—online stochastic gradient for tensor decomposition," in *Proc. Conf. Learning Theory (COLT'15)*, 2015, pp. 797–842.

[55] E. Hazan, K. Y. Levy, and S. Shalev-Shwartz, "On graduated optimization for stochastic non-convex problems," in *Proc. Intl. Conf. Mach. Learning (ICML'16)*, 2016, pp. 1833–1841.

[56] E. Hazan, S. Kale, and S. Shalev-Shwartz, "Near-optimal algorithms for online matrix prediction," *SIAM J. Computing*, vol. 46, no. 2, pp. 744–773, 2017.

[57] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proc. Intl. Conf. Mach. Learning (ICML'16)*, 2016, pp. 314–323.

[58] S. J. Reddi, S. Sra, B. Póczos, and A. Smola, "Fast stochastic methods for nonsmooth nonconvex optimization," *arXiv preprint arXiv:1605.06900*, 2016.

[59] H. Kushner, "Asymptotic global behavior for stochastic approximation and diffusions with slowly decreasing noise effects: global minimization via Monte Carlo," *SIAM J. Appl. Math.*, vol. 47, no. 1, pp. 169–185, 1987.

[60] S. B. Gelfand and S. K. Mitter, "Recursive stochastic algorithms for global optimization in $\mathbb{R}^d$," *SIAM J. Control Optim.*, vol. 29, no. 5, pp. 999–1018, 1991.

[61] H. Fang, G. Gong, and M. Qian, "Annealing of iterative stochastic schemes," *SIAM J. Control Optim.*, vol. 35, no. 6, pp. 1886–1907, 1997.

[62] G. Yin, "Rates of convergence for a class of global stochastic optimization algorithms," *SIAM J. Optim.*, vol. 10, no. 1, pp. 99–120, 1999.

[63] J. Maryak and D. Chin, "Global random optimization by simultaneous perturbation stochastic approximation," *IEEE Trans. Autom. Control*, vol. 53, no. 3, pp. 780–783, Apr. 2008.

[64] M. Raginsky, A. Rakhlin, and M. Telgarsky, "Non-convex learning via stochastic gradient Langevin dynamics: A nonasymptotic analysis," in *Proc. Conf. Learning Theory (COLT'17)*, Amsterdam, Netherlands, Jul. 2017, pp. 1674–1703.

[65] M. A. Erdogdu, L. Mackey, and O. Shamir, "Global non-convex optimization with discretized diffusions," in *Proc. Advances in Neural Information Processing Systems (NeurIPS'18)*, 2018, pp. 9671–9680.

[66] B. Shi, W. J. Su, and M. I. Jordan, "On learning rates and Schrödinger operators," *arXiv preprint*, 2020. [Online]. Available: https://arxiv.org/abs/2004.06977

[67] A. Balsubramani, S. Dasgupta, and Y. Freund, "The fast convergence of incremental PCA," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'13)*, 2013, pp. 3174–3182.

[68] P. Jain, C. Jin, S. M. Kakade, P. Netrapalli, and A. Sidford, "Streaming PCA: Matching matrix Bernstein and near-optimal finite sample guarantees for Oja's algorithm," in *Proc. Conf. Learning Theory (COLT'16)*, 2016, pp. 1147–1164.

[69] Z. Allen-Zhu and E. Hazan, "Variance reduction for faster non-convex optimization," in *Proc. Intl. Conf. Mach. Learning (ICML'16)*, 2016, pp. 699–707.

[70] T. Krasulina, "The method of stochastic approximation for the determination of the least eigenvalue of a symmetrical matrix," *USSR Comput. Mathematics and Mathematical Physics*, vol. 9, no. 6, pp. 189–195, 1969.

[71] Z. Allen-Zhu and Y. Li, "First efficient convergence for streaming k-PCA: A global, gap-free, and near-optimal rate," in *Proc. IEEE 58th Annu. Symp. Found. Comput. Sci. (FOCS'17)*, 2017, pp. 487–492.

[72] C. Tang, "Exponentially convergent stochastic k-PCA without variance reduction," in *Proc. Advances in Neural Information Processing Systems (NeurIPS'19)*, 2019, pp. 12 393–12 404.

[73] M. Simchowitz, A. El Alaoui, and B. Recht, "Tight query complexity lower bounds for PCA via finite sample deformed Wigner law," in *Proc. 50th Annu. ACM Symp. Theory Comput.*, 2018, pp. 1249–1259.

[74] P. Yang, C.-J. Hsieh, and J.-L. Wang, "History PCA: A new algorithm for streaming PCA," *arXiv preprint arXiv:1802.05447*, 2018.

[75] H. Raja and W. U. Bajwa, "Distributed stochastic algorithms for high-rate streaming principal component analysis," *arXiv preprint arXiv:2001.01017*, 2020.

[76] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *Proc. Intl. Conf. Machine Learning (ICML'16)*, 2016, pp. 1225–1234.

[77] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J, Machine Learning Res.*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[78] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[79] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *Proc. Intl. Conf. Learning Representations (ICLR'18)*, 2018.

[80] L. Lei, C. Ju, J. Chen, and M. I. Jordan, "Non-convex finite-sum optimization via SCSG methods," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'17)*, 2017, pp. 2348–2358.

[81] Z. Allen-Zhu, "Natasha: Faster non-convex stochastic optimization via strongly non-convex parameter," in *Proc. 34th Intl. Conf. Machine Learning (ICML'17)*, 2017, pp. 89–97.

[82] ——, "Natasha 2: Faster non-convex optimization than SGD," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'18)*, 2018, pp. 2680–2691.

[83] P. L. Bartlett and S. Mendelson, "Rademacher and Gaussian complexities: Risk bounds and structural results," *J. Machine Learning Res.*, vol. 3, no. Nov, pp. 463–482, 2002.

[84] M. Kearns and D. Ron, "Algorithmic stability and sanity-check bounds for leave-one-out cross-validation," *Neural Computation*, vol. 11, no. 6, pp. 1427–1453, 1999.

[85] O. Bousquet and A. Elisseeff, "Stability and generalization," *J. Machine Learning Res.*, vol. 2, pp. 499–526, Mar. 2002.

[86] S. Mukherjee, P. Niyogi, T. Poggio, and R. Rifkin, "Learning theory: Stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization," *Adv. Comput. Math.*, vol. 25, no. 1, pp. 161–193, Jul. 2006.

[87] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan, "Learnability, stability and uniform convergence," *J. Machine Learning Res.*, vol. 11, pp. 2635–2670, Oct. 2010.

[88] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[89] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[90] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Trans. Automatic Control*, vol. 60, no. 3, pp. 601–615, 2014.

[91] ——, "Stochastic gradient-push for strongly convex functions on time-varying directed graphs," *IEEE Trans. Automatic Control*, vol. 61, no. 12, pp. 3936–3947, 2016.

[92] F. Saadatniaki, R. Xin, and U. A. Khan, "Optimization over time-varying directed graphs with row and column-stochastic matrices," *IEEE Trans. Automatic Control*, 2020.

[93] W. Shi, Q. Ling, G. Wu, and W. Yin, "EXTRA: An exact first-order algorithm for decentralized consensus optimization," *SIAM J. Opt.*, vol. 25, no. 2, pp. 944–966, 2015.

[94] C. Xi and U. A. Khan, "DEXTRA: A fast algorithm for optimization over directed graphs," *IEEE Trans. Automatic Control*, vol. 62, no. 10, pp. 4980–4993, 2017.

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

[96] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Trans. Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[97] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *Proc. 51st IEEE Conf. Decision and Control (CDC'12)*, 2012, pp. 5453–5458.

[98] E. Wei and A. Ozdaglar, "On the $O(1/k)$ convergence of asynchronous distributed alternating direction method of multipliers," in *Proc. IEEE Global Conf. Signal and Information Processing*, 2013, pp. 551–554.

[99] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, 2014.

[100] A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, "DQM: Decentralized quadratically approximated alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. 64, no. 19, pp. 5158–5173, 2016.

[101] T. Tatarenko and B. Touri, "Non-convex distributed optimization," *IEEE Trans. Automatic Control*, vol. 62, no. 8, pp. 3744–3757, 2017.

[102] H.-T. Wai, J. Lafond, A. Scaglione, and E. Moulines, "Decentralized Frank–Wolfe algorithm for convex and nonconvex problems," *IEEE Trans. Automatic Control*, vol. 62, no. 11, pp. 5522–5537, 2017.

[103] J. Zeng and W. Yin, "On nonconvex decentralized gradient descent," *IEEE Trans. Signal Process.*, vol. 66, no. 11, pp. 2834–2848, 2018.

[104] L. Li, A. Scaglione, and J. H. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 4, pp. 725–738, 2011.

[105] A. Bertrand and M. Moonen, "Distributed adaptive estimation of covariance matrix eigenvectors in wireless sensor networks with application to distributed PCA," *Signal Process.*, vol. 104, pp. 120–135, 2014.

[106] I. D. Schizas and A. Aduroja, "A distributed framework for dimensionality reduction and denoising," *IEEE Trans. Signal Process.*, vol. 63, no. 23, pp. 6379–6394, 2015.

[107] J. Fan, D. Wang, K. Wang, and Z. Zhu, "Distributed estimation of principal eigenspaces," *Ann. Statist.*, vol. 47, no. 6, pp. 3009–3031, 2019.

[108] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Machine Learning Res.*, vol. 13, pp. 165–202, Jan. 2012.

[109] K. I. Tsianos and M. G. Rabbat, "Efficient distributed online prediction and stochastic optimization with approximate distributed averaging," *IEEE Trans. Signal Inform. Proc. over Netw.*, vol. 2, no. 4, pp. 489–506, 2016.

[110] M. Nokleby and W. U. Bajwa, "Stochastic optimization from distributed, streaming data in rate-limited networks," *IEEE Trans. Signal Inform. Proc. over Netw.*, vol. 5, no. 1, pp. 152–167, Mar. 2019.

[111] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, "Sample size selection in optimization methods for machine learning," *Mathematical Programming*, vol. 134, no. 1, pp. 127–155, 2012.

[112] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *Proc. 52nd Annu. Allerton Conf. Communications, Control, and Computing*, 2014, pp. 850–857.

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

[114] H. Zhang, S. J. Reddi, and S. Sra, "Riemannian SVRG: Fast stochastic optimization on Riemannian manifolds," in *Proc. Conf. Neural Information Processing Systems (NeurIPS'16)*, 2016, pp. 4592–4600.

[115] Z. Allen-Zhu and Y. Yuan, "Improved svrg for non-strongly-convex or sum-of-non-convex objectives," in *Proc. Intl. Conf. Machine Learning (ICML'16)*, 2016, pp. 1080–1089.

[116] H. Sato, H. Kasai, and B. Mishra, "Riemannian stochastic variance reduced gradient algorithm with retraction and vector transport," *SIAM J. Opt.*, vol. 29, no. 2, pp. 1444–1472, 2019.

[117] B. Li, S. Cen, Y. Chen, and Y. Chi, "Communication-efficient distributed optimization in networks with gradient tracking and variance reduction," in *Proc. Intl. Conf. Artificial Intelligence and Statistics (AISTATS'20)*, 2020, pp. 1662–1672.

[118] H. Sun, S. Lu, and M. Hong, "Improving the sample and communication complexity for decentralized non-convex optimization: A joint gradient estimation and tracking approach," *arXiv preprint arXiv:1910.05857*, 2019.

[119] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. Intl. Conf. Mach. Learning (ICML'18)*, 2018.

[120] A. Krizhevsky, "Learning multiple layers of features from tiny images," Department of Computer Science, University of Toronto, Tech. Report, 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

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

[122] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Intl. Conf. Machine Learning (ICML'15)*, 2015, pp. 1737–1746.

[123] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[124] 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. Conf. Neural Information Processing Systems (NeurIPS'17)*, 2017, pp. 1509–1519.

[125] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," in *Proc. Intl. Conf. Machine Learning (ICML'18)*, 2018, pp. 560–569.