Distributed Stochastic Optimization and Learning

Ohad Shamir[1] and Nathan Srebro[2]

***Abstract*— We consider the problem of distributed stochastic optimization, where each of several machines has access to samples from the same source distribution, and the goal is to jointly optimize the expected objective w.r.t. the source distribution, minimizing: (1) overall runtime; (2) communication costs; (3) number of samples used. We study this problem systematically, highlighting fundamental limitations, and differences versus distributed consensus problems where each machine has a different, independent, objective. We show how the best known guarantees are obtained by an accelerated mini-batched SGD approach, and contrast the runtime and sample costs of the approach with those of other distributed optimization algorithms.**

I. Introduction

Stochastic optimization considers the problem of optimizing the expectation of a stochastic function, which can be written in the form

$$\min_{w\in\mathbf{W}} F(w) \;=\; \mathbb{E}_{z\sim\mathcal{D}}\left[f(w,z)\right], \tag{1}$$

given only f and a finite i.i.d. sample $z_1, z_2, \ldots,$ from an unknown distribution $\mathcal{D}$. In particular, this naturally models most stochastic supervised learning problems: Given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ of labeled examples sampled i.i.d. from an unknown distribution $\mathcal{D}$, and a loss function $\ell(w; \mathbf{x}, y)$ quantifying the performance of a predictor w with respect to an example $(\mathbf{x}, y)$, learning consists of finding a predictor which minimizes the risk $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\ell(w;\mathbf{x},y)\right]$.

In this paper, we focus on convex problems where $\mathbf{W}$ is a convex subset of a vector space and f is convex function over the space. For supervised learning problems, this corresponds to learning a linear predictor (over some possibly implicit) feature space) in a predictor class $\mathbf{W}$ minimizing a convex loss ℓ. For example, we can perform squared-loss regression using $\ell(w;\mathbf{x},y) = (\langle \mathbf{x}, w\rangle - y)^2$, perform logistic regression using $\ell(w;\mathbf{x},y) = \log(1+\exp(-y\langle \mathbf{x}, w\rangle))$, solve linear support vector machines using $\ell(w;\mathbf{x},y) = \max\{0, 1 - y\langle \mathbf{x}, w\rangle\}$, etc.

Over the past few years, using stochastic optimization algorithms for solving learning problems has become increasingly popular. Such methods are relatively easy to implement and scale well to large datasets where sophisticated, second order batch methods (e.g. interior point methods), are impractical. Furthermore, in the "data laden regime", i.e. when data is plentiful and the bottleneck is runtime, first-order stochastic methods dominate sophisticated batch methods both theoretically and in practice [5], [21], [14]. In fact, as we discuss later in more detail, such stochastic methods are in a sense "optimal" for convex learning problems: Their worst-case runtime is essentially equal to the runtime required to just *read* the data using any batch method.

In this paper, we focus on stochastic learning and optimization in a *distributed* setting, and in particular, when the training data to be learned from is distributed across several different machines with limited communication channels between them. This setting has received much interest in recent years, as large-scale datasets and distributed computing platforms are becoming increasingly common. Compared to the serial (single-machine) setting, the distributed setting can be seen as both a challenge and an opportunity: On one hand, computation can be parallelized, leading to potential runtime speedups. On the other hand, since the data is distributed and can't simply be migrated between machines, one may require different algorithmic approaches suited to this setting.

In the distributed context we will consider three resources: sample complexity (number of draws from $\mathcal{D}$), communication costs, and runtime. Ideally, and optimal algorithm for this problem should satisfy three desiderata:

1) *Same statistical performance as the serial setting:* The amount of training data required to solve the stochastic optimization problem, to some given accuracy, should be similar to the serial setting.
2) *Small communication cost*: Since communication is often a dominant bottleneck in distributed computation, we wish to minimize it as much as possible, ideally requiring each machine to broadcast only a constant number of vectors.
3) *Linear runtime speedup*: If we have m machines rather than a single machine, then ideally we would like the runtime to decrease by a factor of m.

Unlike the serial setting, where we already have essentially optimal algorithms for stochastic convex optimization, the situation in the distributed setting is still far from clear. We would like to understand whether such an ideal optimal algorithm exists, or obtain near-ideal algorithms and matching lower bounds for distributed optimization that help us understand how close to this ideal we might hope to get.

And so, in the rest of the paper, we rigorously study the attainable performance for distributed stochastic optimization and learning. We study several approaches, many based on existing methods, with the results summarized in Table I. We show how the best known guarantees are obtained by a mini-batched accelerated stochastic gradient descent (SGD) algorithm with very large mini-batches, and compare

[1]Ohad Shamir is with the department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel ohad.shamir@weizmann.ac.il

[2]Nathan Srebro is with the Toyota Technological Institute at Chicago, Chicago IL, USA and with the Faculty of Computer Science, Technion, Haifa, Israel nati@ttic.edu

the runtime and sample costs of the approach with those of other distributed algorithms. However, perhaps the main take-home message is that we are still far from having an algorithm achieving the trifecta ideal of optimal sample size, communication and runtime speedups.

II. Serial Stochastic Optimization

As background, and in order to establish a basis on top of which we can understand the benefits of challenges of a distributed setting, we begin by reviewing the problem of stochastic optimization in the serial setting, deferring treatment of the distributed setting to the next section.

As discussed in the introduction, we are interested in problems of the form (1) given access to i.i.d. samples $z_1, z_2, \ldots \sim \mathcal{D}$, where $\mathbf{W}$ is a subset of a vector space and f is convex in w. We assume full knowledge of f and $\mathbf{W}$, but that the distribution $\mathcal{D}$ is unknown, and the only information about it is through the samples z_i. Based on the samples $z_1, z_2, \ldots, z_n$, we construct a predicted optimizer $\hat{w}(z_1, z_2, \ldots, z_n)$.

In studying algorithms for this problem, there are two resources to keep in mind, namely sample complexity (number of samples required) and runtime. Our goal is to use these resources as sparingly as possibly, to find some ϵ-optimal predictor $\hat{w}$ for which we can guarantee:

$$\mathbb{E}_{z_1,z_2,\ldots}\left[F(\hat{w})\right] \leq \inf_{w\in\mathbf{W}} F(w) + \epsilon \tag{2}$$

where the expectation is over the training sample, and perhaps randomness in the predation process.

In order to discuss the complexity of a *problem* and reason about lower bounds and *optimal* algorithms, we will take the following view: we will impose specific assumptions, or constraints, on the set $\mathbf{W}$ and objective f, and ask for the best possible guarantee we can ensure with these assumptions. That is, the best runtime and sample complexity which ensures (2) under these assumptions. This is the min-max rate of the problem: the best (over all possible predictors) worst-case performance (for the worst distribution $\mathcal{D}$, under specific assumptions).

Euclidean Setting: In particular, we will focus mostly on the unbounded-dimension Euclidean (Hilbert) setting, where we assume $\mathbf{W}$ is a subset of a Euclidean or Hilbert space contained in a ball of radius $\|\mathbf{W}\| = \sup_{\mathbf{w}\in\mathbf{W}} \|w\|_2$ and f is convex and Lipschitz in w with Lipschitz constant L (i.e. $\forall_{w,w',z} |f(w,z) - f(w',z)| \leq L \cdot \|w - w'\|_2$), where $\|\cdot\|_2$ is a Euclidean or ℓ_2 norm. We assume $\|W\|$ and L are known and are interested in guarantees in terms of $\|W\|$ and L but that do not depend on the dimensionality. To avoid direct discussion of dimensionality, when measuring runtime, and later on communication, we consider vector operations as atomic and count the number of vector operations or number of vectors communicated.

Classical approaches such as Vapnik's statistical learning [28] considers only the sample complexity, while assuming unbounded computational resources. Vapnik thus focuses mostly on a Empirical Risk Minimization (ERM), aka **Sample Average Approximation (SAA)**, as a generic approach to the stochastic optimization problem (1): First collect n samples $z_1, \ldots, z_n$ and construct an empirical approximation of the objective: $\hat{F}(w) = \frac{1}{n}\sum_{i=1}^{n} f(w, z_i)$. Then minimize this empirical approximation, arriving at the empirical risk minimizer $\hat{w} = \arg\min_{w\in W} \hat{F}(w)$. This leaves us with the computational problem of minimizing the empirical approximation, which might scale sharply with the number of examples n involved.

An alternative, the **Stochastic Approximation** approach, is to tackle the stochastic optimization problem directly, at each iteration using a sample z_i to make a small, stochastic, improvement to an iterate w. For examples, using *stochastic gradient descent* (SGD) on the problem (1) amounts to performing iterations of the form:

$$\begin{aligned} w^{(t+1)} &\leftarrow \arg\min_{w\in\mathbf{W}} \left\langle \nabla_w f(w^{(t)}, z_t), w \right\rangle + \frac{1}{2\eta_t} \left\| w - w^{(t)} \right\|_2^2 \\ &= P_{\mathbf{W}}\left(w^{(t)} - \eta_t \nabla_w f(w^{(t)}, z_t) \right) \end{aligned} \tag{3}$$

where $P_{\mathbf{W}}(\cdot)$ is a projection into $\mathbf{W}$ and η_t is a step-size. If an independently drawn sample z_i is used at each iteration, SGD guarantees [15], [3] can be used to bound the suboptimality after T iterations. In particular, when optimizing an L-Lipschitz objective over a domain of radius $\|\mathbf{W}\|$, we have that, with an appropriate step size,

$$T = O\left(\frac{L^2 \|\mathbf{W}\|_2^2}{\epsilon^2} \right) \tag{4}$$

iterations are required to ensure a ϵ-suboptimality. Since each iteration uses one training sample (i.e. one independent draw from the source distribution $\mathcal{D}$), the sample complexity is $n = T$. And, as long as the projection operation can be performed efficiently, the total runtime (in terms of number of vector operations) is also linear in T.

Amazingly, this sample complexity matches that of an ERM approach, and also the lower bound on the required sample complexity in the worst case. I.e. any method for solving (1) under these Lipschitz and boundedness assumptions would require at least these many samples. That is, SGD is not only computationally efficient, but also matches the (worst case) sample complexity of any other possible method. And because its runtime is linear in the time required to just read the data set of the minimal required size, its runtime is also optimal, up to a constant factor, and among methods that read the data[1].

In our Euclidean setting we only rely on Lipschitz continuity. But SGD is also optimal, in a similar way, for smooth [1], [26], [24], and for strongly convex objectives (see below). In particular, in the serial setting, in terms of min-max rates worst-case guarantees, additional smoothness assumptions, including assuming all derivatives are bounded (as is the case, e.g. for the logistic loss), does not change the min-max sample complexity of the problem (unless additional assumptions are made [24]). It is important to keep this in

[1]It is actually possible to slightly improve over SGD in some high dimensional regimes if the algorithm is allowed to only look at a few coordinates of sample points z_i, instead of reading entire vectors[11]

	Samples	Rounds	Communication	Runtime
Ideal Solution	$n(\epsilon)$	O(1)	O(m)	$n(\epsilon)/m$
Centralize	$n(\epsilon)$	1	$n(\epsilon)$	$n(\epsilon)$
Local	$m \cdot n(\epsilon)$	-	0	$n(\epsilon)$
Average-at-the-End	$m \cdot n(\epsilon)$	1	m	$n(\epsilon)$
Hot-Potato SGD	$n(\epsilon)$	m	m	$n(\epsilon)$
SAA: GD	$n(\epsilon)$	$\|\mathbf{W}\| \sqrt{n(\epsilon)}$	$m \|\mathbf{W}\| \sqrt{n(\epsilon)}$	$n(\epsilon)\left(\frac{\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}}{m}\right)$
SAA: Accelerated GD	$n(\epsilon)$	$\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}$	$m\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}$	$n(\epsilon)\left(\frac{\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}}{m}\right)$
SAA: DANE	$n(\epsilon)$	$\|\mathbf{W}\|^2 m$	$\|\mathbf{W}\|^2 m^2$	$> n(\epsilon) \|\mathbf{W}\|^2$
Acc. Mini-Batches	$n(\epsilon)$	$\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}$	$m\sqrt{\|\mathbf{W}\|}\sqrt[4]{n(\epsilon)}$	$n(\epsilon)/m$

TABLE I

SUMMARY OF RESOURCES REQUIRED BY THE VARIOUS APPROACHES STUDIED IN THIS PAPER FOR $\Theta(1)$-LIPSCHITZ, $\Theta(1)$-SMOOTH PROBLEMS, IGNORING LOG-FACTORS.

mind, as in the distributed setting we will occasionally need to rely on higher order smoothness.

The optimality of SGD as discussed above is for a Euclidean setting, where guarantees are based on the domain $\mathbf{W}$ being contained in an ℓ_2 ball, and of the Lipschitz constant (or other higher order derivatives) being measured with respect to an ℓ_2 norm. What happens if the domain $\mathbf{W}$ is very different from a Euclidean ball? Or equivalently, when we measure Lipschitz continuity, smoothness and strong convexity with respect to a different, non-Eucliean, norm? In such cases we must replace the squared Euclidean distance in our derivation (3) of SGD with a different divergence function, resulting in *stochastic mirror descent*. E.g., for Lasso-type problems, when $\mathbf{W}$ is an ℓ_1-ball, we can take the divergence to be an entropic divergence, and for matrix problems where $\mathbf{W}$ is a nuclear norm ball, we can also define an appropriate spectral divergence. In fact, in a fairly broad sense, depending on the geometry of the problem, we can choose an appropriate divergence function $D(\cdot||\cdot)$ such that stochastic mirror descent has essentially optimal sample complexity [25], [26]. For the sake of simplicity, in this paper we consider only the Euclidean setting, but we expect the situation to be similar, and for our results to be generalizable also for other geometries.

In contrast to Vapnik's view, our interest here is mostly in the data laden regime, where data is plentiful and perhaps virtually unlimited, and we ask for the minimal amount of runtime required to achieve suboptimality ϵ for the stochastic optimization problem (1). Our model is that we have a button that we can press and in unit time receive an independent draw from the source distribution $\mathcal{D}$. However, as discussed in the previous paragraphs, the stochastic approximation approaches, namely stochastic gradient descent and stochastic mirror descent, are optimal not only in terms of runtime but also in terms of sample complexity, and the runtime of these approaches is linear in their sample complexity. It would thus be convenient for us, when moving on to distributed settings and comparing back to the serial setting, to refer to the min-max sample complexity of a problem, which we denote by $n(\epsilon)$, and remember that when using stochastic approximation, in the serial setting, the runtime is also linear in $n(\epsilon)$. In particular, in the Euclidean setting we have:

$$n(\epsilon) = \Theta\left(\frac{L^2\|\mathbf{W}\|_2^2}{\epsilon^2}\right) \tag{5}$$

Strongly Convex Objectives

If the objective $F(w)$ is λ-strongly convex in addition to being L-Lipschitz, the iteration complexity of SGD becomes (with appropriate step sizes and iterate averaging, [9], [10], [14], [17], [23]):

$$T = O\left(\frac{L^2}{\lambda\epsilon}\right), \tag{6}$$

again matching the best possible sample complexity. It is important to understand the relationship between the two complexities (4) and (6). For any convex Lipschitz objective $F(w)$, we can add a strongly-convex regularizer to obtain a λ-strongly convex regularized objective,

$$\begin{aligned} f_\lambda(w,z) &= f(w,z) + \frac{\lambda}{2}\|w\|^2 \\ F_\lambda(w) &= F(w) + \frac{\lambda}{2}\|w\|^2 = \mathbb{E}\left[f_\lambda(w,z)\right] \end{aligned} \tag{7}$$

Taking $\lambda = \frac{L}{\|\mathbf{W}\|\sqrt{n}}$, we have that for the output $\tilde{w}_\lambda$ of using SGD to optimize $F_\lambda(w)$,

$$\begin{aligned} &F(\tilde{w}_\lambda) \le F_\lambda(\tilde{w}_\lambda)F_\lambda^* + O\left(\frac{L^2}{\lambda n}\right) \\ &\le F^* + \frac{\lambda}{2}\|w^*\|^2 + O\left(\frac{L^2}{\lambda n}\right) \le F^* + O\left(\sqrt{\frac{L^2\mathbf{W}^2}{n}}\right). \end{aligned} \tag{8}$$

That is, the two iteration complexities (4) and (6) are linked. In fact, in a learning context the strongly convex case (6) is often mostly useful when applied to a regularized objective such as F_λ, as in (8), and so for λ that typically depends on the sample complexity n.

III. DISTRIBUTED STOCHASTIC OPTIMIZATION

We now turn to consider the stochastic optimization problem in a distributed setting.

We consider optimizing a stochastic objective of the form (1) using m machines, where each machine $i = 1, \ldots, m$ has access to n_i i.i.d. samples $z_{i,1}, \ldots, z_{i,n_i}$ from the same source distribution $\mathcal{D}$. We generally consider situations where each machine has the same number of samples $n_i = n$ and so the total number of samples used is $N = nm$.

This can be thought of as follows: each machine has a "button" which can be used to generate a random example from $\mathcal{D}$, where there is possibly a cost to pressing this button. Each machine presses its button, does local computation and communicates with other machines. The "button presses" can of course be integrated into the computation, as with a serial stochastic approximation approach, and if multiple rounds of communication are allowed, we might also interleave communication, computation and "button presses".

If thinking about machine learning with a given training set, this corresponds to randomly partitioning the samples among the m machines, such that effectively each machine has an i.i.d. sample from the source distribution. Either way, our aim is to devise and analyze methods that find ϵ-suboptimal solutions (2), while minimizing the sample complexity, the runtime, and/or the communication.

In this paper we focus on the Euclidean setting described in the previous section, and ask for the optimal sample complexity, runtime and communication as a function of only the radius $\|\mathbf{W}\|$ and Lipschitz constant L. We present our results in terms of the serial sample complexity $n(\epsilon)$ given in (5). As discussed in the introduction, ideal parallelization would mean preserving the same sample complexity $N = n(\epsilon)$, a linear speedup in runtime, reducing it to $O(n(\epsilon)/m)$, and as little as possible communication, ideally independent of ϵ or $n(\epsilon)$. Unlike in the serial approach, we will sometime also assume higher derivatives are bounded. To allow this, and formalize our "optimality" question, we might want to assume all higher order derivatives are bounded. This happens, e.g., when learning a generalized linear predictor with an objective of the form:

$$f(w,(x,y)) = h(\langle w, \phi(x)\rangle, y) \tag{9}$$

with $\phi(x)$ bounded and h is a loss function which with all derivatives bounded w.r.t. its first argument.

Distributed Consensus and Other Related Problems

A Sample Average Approximation approach here would correspond to distributed optimization of:

$$\min_{w\in\mathbf{W}} \hat{F}(w) \quad \text{where} \quad \hat{F}(w) = \frac{1}{m}\sum_{i=1}^{m} \hat{F}_i(w) \tag{10}$$

where for each machine $i \in [m]$, $\hat{F}_i(w) = \frac{1}{n}\sum_{j=1}^{n} f(w, z_{i,j})$ is the local empirical approximation on the machine. That is, minimization of an average of m functions, each known to only a single machine. Problems of this form, where we would like to distributedly minimize an average of m local objectives are known as *distributed consensus* problems, and have received much attention recently [6].

As with the serial case, we could combine an optimization guarantee on (10) with a generalization guarantee relating $\hat{F}(w)$ to $F(w)$, and obtain the desired guarantee on the suboptimality of (1). The generalization guarantee involves only the total number of samples N, and so this approach has the same sample total complexity as the standard serial setting—i.e. we do not require additional samples because of parallelization. The flip-side is that, as with the serial approach, high-accuracy in optimizing $\hat{F}(w)$ might not be necessary nor helpful in our stochastic optimization goal of minimizing $F(w)$.

And so, although distributed stochastic optimization could be reduced to decentralized consensus, and decentralized concensus methods could be used in an SAA approach for distributed stochastic optimization (as discussed above, and analyzed in Section VII), the two problems are quite distinct, might have different optimal solutions, and should be studied differently.

First of all, we are interested in a stochastic setting and our goal is to minimize the expectation (1)—we measure success in terms of the sub-optimality of this expected objective, as in (2), and not in terms of the empirical objective. Because of this, and as in the serial case, methods that only very crudely optimize the empirical objective, such as SGD, might actually be preferable, and even optimal.

Second, we do not treat $\hat{F}_i$ as independent, unrelated functions, since they are all determined by data sampled from the same distribution. In particular, as the number of samples increase, the local empirical objectives $\hat{F}_i$ converge to one another. This assumption works to our advantage, should be taken into account in the analysis, and can also be utilized by optimization methods.

This second distinction also sets our setting apart from those of [4] and [18], which consider a stochastic setting where each machine i has access to a *different* source distribution $\mathcal{D}_i$, and the desired outcome is a predictor which is good on the mixture distribution $\frac{1}{m}\sum_{i=1}^{m}\mathcal{D}_i$. When each machine holds a different component of the source distribution, the machines *must* communicate in order to obtain a consensus predictor good on the mixture distribution. In particular, [4] study lower bounds on the amount of such communication. However, in our setting where each machine has access to the entire source distribution, there is no inherit limit to communications and indeed arbitrarily good solutions can be obtained without any communication at all (e.g. with the `Local` approach studied in Section IV). In a sense, learning a mixture source with different components on different machines makes for a *harder* problem than in the standard serial setting (we must both learn and consolidate between the different components), whereas our setting is *easier* than the serial setting—machines can always choose to ignore each other, but by cooperating they can hopefully reduce the required runtime.

Much attention has also been devoted to the network topology in which machines can communicate. In this paper, we limit ourselves to studying the simple case of broadcast communications, aiming to first understand distribution stochastic optimization in this setting before studying the effect of network structure.

IV. Two Baseline Approaches

We now turn to study specific algorithmic approaches for our distributed stochastic problem, starting with two trivial

baseline approaches which do not require any specialized algorithms. We note that our analysis for all methods is summarized in table I.

Perhaps the simplest approach one can take, which we denote as `Local`, is the following: Each machine i independently minimizes the empirical objective $\hat{F}_i(w)$ to obtain $\hat{w}_i = \arg\min \hat{F}_i(w)$, or alternatively just runs a single pass of SGD on its samples, using no communication at all (if we desire a single output vector, we can just output $\hat{w}_1$ with minimal communication cost).

This approach is nonsensical for solving a consensus problem where the goal is to minimize an average of unrelated functions. However, in our stochastic setting, the generalization guarantee based on $n = N/m$ samples applies, and can also be used to obtain a suboptimality guarantee for $\hat{F}(\cdot)$, is desired. E.g. for an L-Lipschitz objective over a bounded domain, we have that $F(\hat{w}_1) \leq F^* + O(\sqrt{\frac{L^2\|\mathbf{W}\|_2^2}{n}})$, and less interestingly also $\hat{F}(\hat{w}_1) \leq \hat{F}(\hat{w}) + O(\sqrt{\frac{L^2\|\mathbf{W}\|_2^2}{n}})$. That is, the total sample complexity of this approach is $N = nm = O(\frac{L^2\|\mathbf{W}\|^2}{\epsilon^2}m)$. This is of course still a useless approach in terms of harnessing the power of parallelization (we are not really parallelizing anything), but the point is that with essentially zero communication we can still obtain an ϵ-suboptimal solution (2).

At the other extreme, if communication is cheap, all the machines can send their data to a central machine, which will then use a standard serial approach. We denote this approach as `Centralize`. Here, there is no increase in sample complexity, $N = n(\epsilon)$, but there is also no reduction in runtime.

The above extreme approaches are not useful approaches to solving the problem (they offer no parallelization), but are important to keep in mind as baseline trivial approach to distributed stochastic optimization: we must measure any non-trivial approach against them to understand its benefit.

The natural question, of course, is whether it is possible to beat these baselines—can we get a reduction in total elapsed runtime? Does it come at a price of increased sample complexity, or is it possible to do so without increasing the total number of samples required? What can be done with limited communication? What are the tradeoffs between communication, sample complexity and elapsed runtime?

V. Average-at-the-End

Another simple distributed optimization approach, with minimal communication complexity, is to perform independent optimization on each machine to obtain $\tilde{w}_i$, either using ERM, $\tilde{w}_i = \arg\min_w \hat{F}_i(w)$, or using SGD on $z_{i,1}, \ldots, z_{i,n}$, and then return the averaged result:

$$\bar{w} = \frac{1}{m}\sum_{i=1}^{m} \tilde{w}_i. \tag{11}$$

This approach was studied in [30] and rigorously analyzed in [29]. When ERM is used and $F(w)$ is λ-strongly convex, and $f(w,z)$ is L-Lipschitz, H-smooth and has a J-Lipschitz Hessian, [29] obtain a guarantee on $\bar{w}$ of the following form (in expectation over the samples):

$$\mathbb{E}\left[\|\bar{w} - w^*\|^2\right] \leq \frac{L^2}{\lambda^2 nm} + O\left(\frac{L^4J^2}{\lambda^6 n^2}\right) + \text{lower order terms} \tag{12}$$

When SGD is used instead of ERM, a similar guarantee is also provided with slightly stronger assumptions. When λ is taken to be fixed, and $n \to \infty$, even if m increases with n, the first term in (12) is the dominant term. Hence we get a sample complexity which matches (to leading order) the serial sample complexity. However, as we discussed earlier, in many learning applications strong convexity is a result of regularization, and λ should decrease with N, or with the desired accuracy, usually at a rate of roughly $1/\sqrt{N}$. With this scaling in mind, the benefit of (12) is unclear. Furthermore, the guarantee (12) is only on the distance to the optimum $w^* = \arg\min_w F(w)$. In learning, and other stochastic optimization problems, we are interested in the suboptimality of the objective. What can be ensured in this regard from (12) is:

$$F(\bar{w}) - F^* \leq \frac{HL^2}{\lambda^2 nm} + O\left(\frac{HL^4J^2}{\lambda^6 n^2}\right) + \text{lower order terms.} \tag{13}$$

Comparing (13) with (6) we see an extraneous factor of H/λ even in the first term, on top of the additional λ terms in the second term. Optimizing over λ, the best that can be ensured from (13) for learning problems requiring regularization is therefor only a sample complexity that scales as $1/\epsilon^3$ rather then $1/\epsilon^2$. If ERM is used on each machine [29] also suggested a bias-corrected approach that reduced the dependence on n in the second term to $1/n^3$, rather then $1/n^2$, but the problematic dependence on λ remains.

These deficiencies are not only in the analysis. Consider (11) with $\tilde{w}_i = \hat{w}_i$ and let $\hat{w}$ be the true ERM of the combined $\hat{F}(w)$ (using $N = nm$ examples). As shown in [22], it is possible to construct, for any $n > 9$ and $\lambda < \frac{1}{9\sqrt{n}}$, a one dimensional stochastic optimization problem, with bounded first, second and third derivatives, such that in expectation over the samples,

$$\|\bar{w} - w^*\| = \Omega\left(\frac{1}{\lambda^2 n}\right)$$

and

$$F(\bar{w}) - F^* = \Omega\left(\frac{1}{\lambda^2 n}\right),$$

yet it is guaranteed [19] that $\mathbb{E}\left[\|\hat{w} - w^*\|\right] = O(\frac{1}{\lambda N})$ and $F(\hat{w}) - F^* = \Omega(\frac{1}{\lambda N})$ (recall that $N = nm$). That is, in this example (in which λ is small) we do not gain at all from averaging over multiple machines, and obtain the same performance as the trivial baseline working on only the $n = N/m$ samples available to a single machine.

The bottom line is that the average-at-the-end approach does not give us any benefit over the baseline `Local` approach.

VI. Hot-Potato SGD

Is it possible to at least match the sample complexity and communication guarantees of the two baseline approaches?

I.e. to at least have a method with minimal communication (in particular, that does not communicate the entire data set), and optimal sample complexity?

To do so, we can simulate a serial stochastic optimization algorithm over the distributed data, utilizing the fact that they all come from the same distribution: The first machine starts at some $w^{(1,0)}$ and runs one pass of stochastic approximation (e.g. stochastic gradient descent) on its n examples $z_{1,1}, \ldots, z_{1,m}$, arriving with the iterate $w^{(1,n)}$. It then passes $w^{(1,n)}$ to the second machine, which runs a pass of the stochastic approximation method starting from this iterate, using its n examples, passing it along to the third machine. Each machine i thus receives the iterate $w^{(i-1,n)}$ from machine $i-1$, uses it to initialize $w^{(i,0)} = w^{(i-1,n)}$, runs a pass of n stochastic approximation updates using its n independent samples, and then passes the iterate $w^{(i,n)}$ on to machine $i+1$. If necessary, the average iterate can also be passed along. The end result is that $w^{(m,n)}$ is exactly an iterate after nm independent stochastic approximation updates, and has the same expected suboptimality as when using N examples on a single machine. We thus used a total of $O(m)$ communication, almost the minimum we can expect, and maintained a total sample complexity of $N = n(\epsilon)$. The total work done on all machines (total number of operations performed) is also the best we can expect, i.e. $O(n(\epsilon))$. However, the total elapsed runtime (wall time) is $O(mn(\epsilon))$, even if we ignore communication time. That is, we pay no cost in terms of sample complexity, and almost no communication cost, but receive no benefit in terms of elapsed runtime.

VII. SAA Approaches

As mentioned already in Section III, SAA can be implemented as a distributed consensus problem. In this Section, we study such approaches, were distributed consensus methods are applied to the empirical objective (10).

As in studying SAA in the serial case, in order to understand distributed consensus in the context of stochastic optimization, we must think of the expected objective $F(w)$ and the effect of approximately optimizing the empirical objective on $F(w)$. In particular, we must combine the optimization guarantee with a generalization guarantee relating $\hat{F}(w)$ to $F(w)$. In particular, in the Euclidean setting, in order to ensure ϵ-suboptimality of the expected objective, as in (2), we must optimize $\hat{F}(w)$ to within $O(\epsilon)$ suboptimality, and use at least $N = nm = O(n(\epsilon))$ samples. We will thus study the runtime and communication costs of empirical minimization of $\hat{F}(w)$ to within ϵ, with $n = n(\epsilon)/m$ samples per machine.

One simple such approach is simulating (standard, deterministic) gradient descent over the data: In each iteration t, all machines work on the same iterate $w^{(t)}$, and each machine computes its local gradient $\nabla \hat{F}_i(w^{(t)})$. These are then averaged across machines, requiring $O(m)$ communication per iteration, to obtain the overall gradient $\nabla \hat{F}(w^{(t)})$. Each machines can then compute the next iterate

$$w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla \hat{F}(w^{(t)}) \tag{14}$$

for a suitable pre-determined step size η_t (performing distributed line search is also possible, but not necessary in order to achieve optimal worst-case behavior).

For non-smooth objectives, we do not get any improvement over the baseline, as $n(\epsilon)$ iterations are required. But if the objective is H-smooth (i.e. has H-Lipschitz continuous gradients), then $O\left(H \left\|\mathbf{W}\right\|^2 / \epsilon\right)$ iterations of gradient descent are sufficient to get an optimization error of ϵ. Therefore, in our distributed implementation, $O\left(H \left\|\mathbf{W}\right\|^2 / \epsilon\right)$ communication rounds are sufficient to get to a similar optimization error.

Furthermore, for a smooth objective, instead of standard gradient descent, one can use Nesterov's *accelerated* gradient descent [16], which makes more efficient use of the computed gradients, at essentially the same computational cost. In a nutshell, instead of updating as in Equation 14, one maintains an auxiliary vector $v^{(t)}$, and performs the update

$$\begin{aligned} w^{(t+1)} &\leftarrow v^{(t)} - \eta \nabla \hat{F}(v^{(t)}) \\ v^{(t+1)} &\leftarrow (1-\gamma_t) w^{(t+1)} + \gamma_t w^{(t)}, \end{aligned}$$

using suitable parameters η, γ_t. Again, each machine can maintain the iterates w and v, communicate so as to calculate $\nabla \hat{F}(v^{(t)})$ jointly, and then perform the update independently. Compared to regular gradient descent, the computational and communication cost per iteration is roughly the same. However, the number of required iterations is only $\sqrt{H \left\|\mathbf{W}\right\|_2^2 / \epsilon}$. For $L, H = \Theta(1)$, expressed in terms of $n(\epsilon)$, we thus need only $O\left(\sqrt{\left\|\mathbf{W}\right\|_2} \sqrt[4]{n(\epsilon)}\right)$ iterations, with an associated amount of communication of $O\left(m\sqrt{\left\|\mathbf{W}\right\|_2} \sqrt[4]{n(\epsilon)}\right)$. In terms of runtime, each gradient computation requires $n(\epsilon)/m$ elapsed runtime (each machine requires a linear scan over its $n_i = n(\epsilon)/m$ local training points), yielding $O\left(\frac{n(\epsilon)}{m} \sqrt{\left\|\mathbf{W}\right\|_2} \sqrt[4]{n(\epsilon)}\right)$ overall runtime.

With only a single machine, such an SAA approach is, as we know, sub-optimal. The sample complexity is the same as SGD, but the runtime is greater by a factor of $\sqrt{\left\|\mathbf{W}\right\|_2} \sqrt[4]{n(\epsilon)}$—even assuming smoothness and using acceleration, we cannot gain by performing SAA in the serial setting. But in the distributed setting, such an SAA approach can be beneficial in certain regimes with a large number of machines. In particular, when $m < \Omega\left(\sqrt{\left\|\mathbf{W}\right\|_2} \sqrt[4]{n(\epsilon)}\right)$, we *do* get a runtime improvement over serial SGD. But the runtime here is still not quite a factor of m improvement as we might want, and it comes at a significant communication cost.

We can also consider SAA with more sophisticated distributed consensus algorithms. E.g., a popular approach uses the Alternating Direction Method of Multipliers, or ADMM [6]. In the context of distributed optimization, it relies on each machine i iteratively computing a solution $w_i^{(t)}$ to a local optimization problem of the form

$$\min_{w \in \mathbf{W}} \hat{F}_i(w) + \left\langle u^{(t-1)}, w \right\rangle + \frac{\rho}{2} \left\| w - w^{(t-1)} \right\|_2^2,$$

where ρ is a suitable constant and $u^{(t-1)}$ is a vector depending on $w_i^{(t-1)}$ and $w^{(t-1)} = \frac{1}{m}\sum_{i=1}^m w_i^{(t-1)}$. Thus, the machines alternate between a local optimization phase, and a communication phase where the solutions are shared and their averages used for the next iteration. This approach is potentially problematic, as it requires us to solve a full-blown optimization problem at each iteration. Although in many cases one can get a good solution after a small number of iterations, the best worst-case guarantee we are familiar with is $O((1/\lambda)\log(1/\epsilon))$ iterations for strongly convex and smooth functions [12], [13]. This is no better than the iteration bound for gradient descent, and therefore the runtime cost is much worse.

Recently [22] proposed a different approach, denoted as DANE, which resembles ADMM in its architecture but has better performance guarantees. Like ADMM, DANE requires each machine to iteratively compute a solution w_i^t to a local optimization problem, followed by a communication phase. However, the optimization problem is slightly different, of the form

$$\min_{w\in\mathbf{W}} \hat{F}_i(w) - \left\langle \nabla\hat{F}_i(w^{(t-1)}) - \eta\nabla\hat{F}(w^{(t-1)}), w\right\rangle + \frac{\mu}{2}\left\|w - w^{t-1}\right\|_2^2$$

for suitable constants η, μ. For λ-strongly convex quadratic objectives, the number of required iterations to reach optimization error ϵ is

$$O\left(\frac{1}{\lambda^2 n}\log(dm)\log(1/\epsilon)\right), \tag{15}$$

where d is the dimension and n is the number of examples per machine. Moreover, since any strongly convex and smooth function is locally quadratic, this bound should also characterize the convergence of the algorithm more generally. The important point here is that DANE takes advantage of the relatedness of the local objectives, and indeed the iterations complexity (15) *decreases* as $n \to \infty$.

To convert Equation 15 to a bound on non-strongly convex functions, we can simply use explicit regularization with λ on the order of $1/\sqrt{\|\mathbf{W}\|_2^2 nm}$, and get an iteration bound of $O\left(\|\mathbf{W}\|_2^2 m\log(1/\epsilon)\right)$. As a result, the amount of communication required by the algorithm is $O\left(\|\mathbf{W}\|_2^2 m^2\log(1/\epsilon)\right)$. Remarkably, this bound is independent of the data size, in sharp contrast to the other methods discussed in this section. Thus, at least in terms of communication, this method is rather efficient. Its main disadvantage is runtime, since it still requires each machine to solve an optimization problem every round. Since this requires at least one pass over each machine's $n(\epsilon)/m$ data points, a crude lower bound on the required runtime is $O\left(\|\mathbf{W}\|_2^2 m\log(1/\epsilon)(n(\epsilon)/m)\right) = \tilde{O}\left(n(\epsilon)\|\mathbf{W}\|_2^2\right)$, which is already inferior to some of the methods discussed earlier.

VIII. Mini-Batch SGD

A compromise between standard SGD and deterministic gradient descent on the empirical objective is mini-batched SGD. Instead of using a single sample point in each iteration t of a stochastic approximation method such as SGD, we might consider using a "mini-batch" of b independent samples $z_{t,(1)},\ldots,z_{t,(b)}$ and replacing the gradient estimate $\nabla_w f(w,z_t)$ with the better estimate $\frac{1}{b}\sum_{i=1}^b \nabla_w f(w,z_{t,(i)})$. This gradient computation can then be parallalized, with each machine computing and averaging gradients on b/m independent sample points. For Lipschitz objectives over a bounded domain, using mini-batches does not actually reduce the *worst case* number of iterations required compared to standard SGD, and thus increases the sample complexity without any possible worst-case improvement in parallel runtime, even though such gains are clearly observed in practice [20]. Recently, two alternative conditions that *do* ensure parallelization speedups from mini-batches have been established. Here, we focus on the case of smooth objectives.

If the objective is H-smooth, then as long as we use mini-batches of size $b \le b_{\max} = O(L^2/(H\epsilon)) = O((L/H\|\mathbf{W}\|)\sqrt{n(\epsilon)})$, the number of required iterations decreases linearly with $1/b$, and we can ensure an ϵ-suboptimality of (1) without increasing the (worst case) total sample complexity by more then a factor of two [8], [2], [1]. By using an *accelerated* variant of mini-batched gradient descent, with the same cost per iteration, we can use even larger mini-batches of size of up to $b_{\max} = O\left(L^{3/2}\|\mathbf{W}\|/(H^{1/4}\epsilon^{3/2})\right) = O\left(n(\epsilon)^{3/4}/H^{1/4}\|\mathbf{W}\|^{1/2})\right)$ while maintaining a linear decrease in the number of iteration and thus without increasing the sample complexity [7].

How can we implement mini-batched (accelerated) gradient descent in a distributed setting? One naive approach would be to process, at each iteration, one sample point on each machine, averaging their gradients across machines, and thus using a mini-batch of size $b = m$. As long as $m = b \le b_{\max}$, and assuming no overhead, we would indeed enjoy a linear speedup. But the communication costs in this case would be prohibitive, as we would be effectively communicating the entire data set (or rather the gradients for the entire data set). Furthermore, the overhead of performing the distributed averaging of the gradients would likely outweight any runtime gains.

What we should do is use a minibatch size larger then the number of machines, and so average multiple gradients on the same machine, and communicate only this average. Using a mini-batch of size b and $m < b$ machines, we would process b/m examples on each machine at each iteration, communicate the averaged gradients obtained on each machine, and then average these average gradients, for a total communication cost of $O(m)$ per iteration. As long as $b < b_{\max}$ the overall sample complexity $N = O(n(\epsilon))$ is maintained (up to perhaps a factor of two), the number of iterations performs is $n(\epsilon)/b$, and so the runtime (number of examples processed on each machine) is reduced linearly to $n(\epsilon)/m$, while the communication costs are $\frac{n(\epsilon)}{b}\cdot m$, corresponding to communicating m vectors at each of $n(\epsilon)/b$ iterations. The runtime and sample complexity are thus

unaffected by the mini-batch size (as long as it is small enough), while communication decreases linearly with the size of the mini-batch. It is thus important to use the largest possible mini-batch size that does not hurt the sample complexity, i.e. $b = b_{\max}$. In particular, we see here that the main benefit of acceleration is not in reducing runtime, but rather in allowing larger mini-batches and thus reducing communication costs.

Overall, when $L, H = \Theta(1)$, using acceleration and a mini-batch size of $b = b_{\max} = O\left(\|\mathbf{W}\|/\epsilon^{3/2}\right) = O\left(n(\epsilon)^{3/4}/\|\mathbf{W}\|^{1/2})\right)$, yields overall runtime of $n(\epsilon)/m$ (a linear speedup) and a communication cost of $O\left(m\sqrt{\|\mathbf{W}\|_2}\sqrt[4]{n(\epsilon)}\right)$, and no deterioration in sample complexity as long as $m \ll b_{\max}$.

Compared to an SAA approach using deterministic accelerated gradient descent, we get the same iteration complexity and communication cost, but with a significantly reduced runtime, avoiding the factor of $\sqrt{\|\mathbf{W}\|_2}\sqrt[4]{n(\epsilon)}$, and thus displaying a linear speedup.

We note that even for non-smooth objectives, an alternate condition on the spectral norm of the data can also yield similar speedups, with a mini-batch size that depends on the spectral norm [27].

IX. CONCLUSION

In this paper we rigorously defined and studied the problem of distributed stochastic optimization, emphasizing that the methods achieving optimal, or even non-trivial, performance for the problem are different than either the methods used in the serial setting, or those that are good for generic distributed consensus problems. Of the methods considered, we the best guarantee is obtained by accelerated mini-batch SGD with a maximally possible mini-batch size. For this method, and up to a generous limit on the number of machines, we obtain linear speedups with no deterioration in sample complexity, and with a communication cost that scales as $\sqrt[4]{n(\epsilon)}$. I.e., the amount of data to be communicated scales as the fourth root of the total data set size. Although such amounts of communication might often be reasonable, it still remains to understand whether this is the minimum possible amount of communication, or whether other methods exist which require even less communication. In particular, the question of the existence of an "ideal" method, with linear runtime speedup and requiring only a constant amount of communication from each machine, is still open.

ACKNOWLEDGMENTS

This work was supported by Intel (ICRI-CI). OS was supported in part by an Israel Science Foundation grant (No. 425/13) and a Marie-Curie Career Integration Grant, NS was supported in part by a Google Research Award and by the National Science Foundation.

REFERENCES

[1] An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1-2):365–397, 2012.

[2] A. Agarwal and J. Duchi. Distributed delayed stochastic optimization. In *NIPS*, 2011.

[3] F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *NIPS*, 2011.

[4] M. Balcan, A. Blum, S. Fine, and Y. Mansour. Distributed learning, communication complexity, and privacy. In *COLT*, 2012.

[5] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *NIPS*, 2007.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[7] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *NIPS*, 2011.

[8] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *J. of Machine Learning Research*, 13:165–202, 2012.

[9] E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[10] E. Hazan and S. Kale. Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. In *COLT*, 2011.

[11] E. Hazan, T. Koren, and N. Srebro. Beating SGD: Learning svms in sublinear time. In *NIPS*, 2011.

[12] M. Hong and Z.-Q. Luo. On the linear convergence of the alternating direction method of multipliers. *arXiv:1208.3922*, 2012.

[13] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem. Explicit convergence rate of a distributed alternating direction method of multipliers. *arXiv:1312.1085*, 2013.

[14] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM J. on Optimization*, 19(4):1574–1609, 2009.

[15] A. Nemirovski and D. Yudin. *Problem complexity and method efficiency in optimization*. Nauka Publishers, Moscow, 1978.

[16] Y. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[17] A. Rakhlin, O. Shamir, and K. Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. In *ICML*, 2012.

[18] S. Ram, A. Nedić, and V. Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

[19] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Stochastic convex optimization. In *COLT*, 2009.

[20] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[21] S. Shalev-Shwartz and N. Srebro. SVM optimization: inverse dependence on training set size. In *ICML*, 2008.

[22] O. Shamir, N. Srebro, and T. Zhang. Communication efficient distributed optimization using an approximate newton-type method. In *ICML*, 2014.

[23] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *ICML*, 2013.

[24] N. Srebro, K. Sridharan, and A. Tewari. Smoothness, low noise and fast rates. In *NIPS*, 2010.

[25] N. Srebro, K. Sridharan, and A. Tewari. On the universality of online mirror descent. In *NIPS*, 2011.

[26] K. Sridharan. Learning from an optimization viewpoint. *PhD Thesis, Toyota Technological Institute at Chicago*, 2012.

[27] M. Takác, A. Bijral, P. Richtárik, and N. Srebro. Mini-batch primal and dual methods for svms. In *ICML*, 2013.

[28] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[29] Y. Zhang, J. Duchi, and M. Wainwright. Communication-efficient algorithms for statistical optimization. *J. of Machine Learning Research*, 14(1):3321–3363, 2013.

[30] M. Zinkevich, M. Weimer, L. Li, and A. Smola. Parallelized stochastic gradient descent. In *NIPS*, 2010.