INVITED PAPER

# Stochastic Quasi-Newton Methods

*This article discusses recent developments to accelerate convergence of stochastic optimization through the exploitation of second-order information and shows applications in the context of predicting the click-through rate of an advertisement displayed in response to a specific search engine query.*

By Aryan Mokhtari, *Member IEEE*, and Alejandro Ribeiro, *Member IEEE*

**ABSTRACT** | Large-scale data science trains models for data sets containing massive numbers of samples. Training is often formulated as the solution of empirical risk minimization problems that are optimization programs whose complexity scales with the number of elements in the data set. Stochastic optimization methods overcome this challenge, but they come with their own set of limitations. This article discusses recent developments to accelerate the convergence of stochastic optimization through the exploitation of second-order information. This is achieved with stochastic variants of quasi-Newton methods that approximate the curvature of the objective function using stochastic gradient information. The reasons for why this leads to faster convergence are discussed along with the introduction of an incremental method that exploits memory to achieve a superlinear convergence rate. This is the best-known convergence rate for a stochastic optimization method. Stochastic quasi-Newton methods are applied to several problems, including prediction of the click-through rate of an advertisement displayed in response to a specific search engine query by a specific visitor. Experimental evaluations showcase reductions in overall computation time relative to stochastic gradient descent algorithms.

**KEYWORDS** | Optimization algorithms; quasi-Newton methods; stochastic optimization.

## I. INTRODUCTION

Convex statistical risk minimization (SRM) problems involve determination of an optimal parameter $\mathbf{w}^* \in \mathbb{R}^p$ relative to a convex cost $f(\mathbf{w}, \boldsymbol{\Theta})$ that is averaged over the probability distribution of the random variable $\boldsymbol{\Theta}$:

$$\mathbf{w}_S^* := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} F(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \mathbb{E}_{\boldsymbol{\Theta}}[f(\mathbf{w}, \boldsymbol{\Theta})]. \tag{1}$$

The problem in (1) is statistical in that we are minimizing a cost averaged over a distribution. A close relative of (1) is empirical risk minimization (ERM), in which we are given $N$ samples $\boldsymbol{\theta}_i$ and the goal is to find the minimum cost averaged over the given samples:

$$\mathbf{w}^* := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} f(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w}, \boldsymbol{\theta}_i). \tag{2}$$

The statistical problem in (1) can be interpreted as an alternative writing of (2) if we consider the samples $\boldsymbol{\theta}_i$ to be drawn uniformly at random with probability $1/N$. More common and more interesting is to think of the samples $\boldsymbol{\theta}_i$ as drawn from the random variable $\boldsymbol{\Theta}$. In this case, (2) is an approximation of (1) if $N$ is sufficiently large and the cost functions satisfy mild regularity conditions.

Solving (1) or (2) comes with a unique challenge that is the difficulty of computing descent directions. For instance, computing gradients of $f(\mathbf{w})$ in the ERM problem in (2) requires that we compute individual gradients of each of the $N$ functions $f(\mathbf{w}, \boldsymbol{\theta}_i)$, an operation with prohibitive cost when $N$ is large. In the SRM problem in (1), the cost of computing gradients is compounded with the cost of acquiring the probability distribution of the data set. Yet, it is clear that the acquisition of this distribution, as well as the large computational costs of evaluating gradients, is unnecessary. Both the gradients of $F(\mathbf{w})$ in (1) and $f(\mathbf{w})$ in (2) can be approximated by (sub)sampling a batch of data points $\boldsymbol{\theta}_i$. In fact, any batch (sub)sampling of data points $\boldsymbol{\theta}_i$ yields stochastic gradients characterized by the fact that their expected values are either a gradient of $F(\mathbf{w})$ or $f(\mathbf{w})$, depending on the problem that we are considering.

Manuscript received January 29, 2020; revised April 14, 2020, August 17, 2020, and September 8, 2020; accepted September 8, 2020. Date of publication September 28, 2020; date of current version October 27, 2020. This work was supported in part by the National Science Foundation (NSF) under Grant CCF-2007668. The work of Alejandro Ribeiro was supported by NSF HDR TRIPODS under Grant #1934960. *(Corresponding author: Aryan Mokhtari.)*

**Aryan Mokhtari** is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: mokhtari@austin.utexas.edu).

**Alejandro Ribeiro** is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: aribeiro@seas.upenn.edu).

Digital Object Identifier 10.1109/JPROC.2020.3023660

This unbiasedness of stochastic gradients indicates that, on average, they point in the right direction. It is, therefore, reasonable to expect convergence toward the optimal argument of a stochastic gradient descent (SGD) method in which we iterate through variable updates in directions determined by corresponding stochastic gradients. This reasonable expectation turns out correct as can be formally established with simple martingale arguments.

At this point, it is important to remark that the interest in SGD has grown in the last decade because of the central role it plays in machine learning. Within this context, the objective is to learn the parameter $\mathbf{w}$ that minimizes the loss $f(\mathbf{w}, \boldsymbol{\Theta})$ over the probability distribution of data entries $\boldsymbol{\Theta}$. This is the SRM problem in (1). The assumption in machine learning is that the probability distribution of the data is unknown, but that we can access samples from the data distribution. This can lead directly to the formulation of an SGD algorithm. Alternatively, we can group available samples to write the ERM problem in (2). This also leads to the formulation of an SGD method. The interest in SGD, therefore, stems from its use in SRM, in general, and ERM, in particular. It is not exaggerated to say that SGD is the workhorse of machine learning.

Such widespread use of SGD is in spite of its slow convergence. It takes large numbers of iterations to approximate $\mathbf{w}^*$. This is not a surprise to any reader familiar with optimization methods or statistical methods. Indeed, deterministic gradient descent converges slowly when the level sets of the objective function are not close to spherical, and in the case of stochastic gradient descent, this is compounded with the need to average out the randomness in gradient computations. Two of the most active research areas of the last decade are concerned with methods to overcome both these limitations (see Section I-A for a review of existing literature).

The goal of this article is to survey the stochastic quasi-Newton methods that have been shown in theory and practice to mitigate the effects of challenging curvature in stochastic optimization. The methods that we will describe are inspired by the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [1]–[4] that is used in deterministic optimization when computing Newton steps is impossible or undesirable. In particular, we will cover the following methods.

*Online BFGS and regularized stochastic BFGS:* Wherever BFGS uses a gradient, online (o) BFGS replaces it with a stochastic gradient. This simple modification creates an algorithm that is experimentally observed to outperform SGD [5]. A drawback of oBFGS is that, in some situations, it generates divergent sequences. This happens because the noise of stochastic gradients can be catastrophically amplified in curvature estimation. Regularized stochastic BFGS (RES) [6] resolves this issue with the incorporation of regularization. We point out that regularizing a quasi-Newton method is not straightforward because the simple addition of a regularizer would eliminate the benefits of curvature estimation. RES chooses a regularizer that is guaranteed to keep the benefits of oBFGS while avoiding the possibility of divergent steps (see Section III).

*Online limited memory BFGS:* Stochastic BFGS inherits from BFGS an iteration cost of order $O(p^2)$, in which RES further scales to order $O(p^3)$. The online limited memory (oL) BFGS algorithm reduces this computational cost [7], [8]. The fundamental idea in stochastic BFGS, RES, and oLBFGS is to continuously estimate the objective's curvature using past stochastic gradients. They differ in that stochastic BFGS and RES use all past stochastic gradients to do so, while oLBFGS uses a fixed moving window of $\tau$ past gradients. This modification, along with careful algebraic manipulations, yields the computational cost of order $O(\tau p)$. Using oLBFGS extends the applicability of stochastic quasi-Newton methods to problems where $\mathbf{w}$ is of high dimension (see Section IV).

*Linearly convergent stochastic LBFGS algorithm:* RES and oLBFGS are successful in expanding the application of quasi-Newton methods to stochastic settings, but their convergence rate is sublinear. This is not better than the convergence rate of SGD and, as is also the case in SGD, is a consequence of the stochastic approximation noise that necessitates the use of diminishing stepsizes. To resolve this issue, there has been a recent line of work on variance reduction techniques for the first-order methods that allow achieving linear rate while using a fixed stepsize [9]–[17]. The stochastic LBFGS (SLBFGS) method proposed in [18] leverages this idea to reduce the noise of gradient estimation in stochastic quasi-Newton methods, leading to a linearly convergent method (see Section V).

*Incremental quasi-Newton:* The use of variance reduction techniques in SGD recovers the linear convergence rate of GD [12]. On the other hand, the use of variance reduction in stochastic quasi-Newton methods leads to a linear convergence rate [18] but does not recover a superlinear rate. Hence, there exists an interesting mismatch, and a natural question that arises is the possibility of a stochastic (incremental) quasi-Newton method that recovers the superlinear convergence rate of deterministic quasi-Newton algorithms. The incremental quasi-Newton (IQN) method proposed in [19] uses variance reduction techniques as well as a consistent Taylor's expansion to attain superlinear convergence. IQN is the only stochastic quasi-Newton method that is known to achieve superlinear convergence (see Section VI).

The methods that we survey in this article offer a set of choices of broad applicability. For problems where the variable dimension is small, RES is a workable alternative. For a larger dimension, oLBGS offers some of the advantages of curvature estimation along with a computational cost that is not significantly larger than the computational cost of regular SGD. For the finite-sum setting, in which it is possible to compute the exact objective gradient once in a while, SL-BFGS is the method of choice as it converges linearly. IQN is the best-performing algorithm in theory and practice. However, the implementation of the variance reduction technique in IQN requires maintaining a history

of past gradients, which may be challenging to store when the number of samples $N$ is large. SLBFGS and IQN are also unworkable for large $p$ as they have a computational cost per iteration that is of order $O(p^2)$. We compare the performance of these methods versus stochastic first-order methods in detail (see Section VII). Future research on stochastic quasi-Newton methods must further contend with the issues of computational cost, memory, and convergence rates. Investigation of applications to nonconvex problems is also of interest (see Section VIII).

### A. Related Literature

Recent studies on stochastic first-order methods include, for example, [9], [10], and [20]–[23]. Hybrid approaches that use both gradients and stochastic gradients or update descent directions so that they become progressively closer to gradients have been proposed recently [10], [14], [15]. The incremental aggregated methods, which use memory to aggregate the gradients of functions, are successful in reducing the noise of gradient approximation to achieve linear convergence rate [9]–[12]. The work in [9] suggests a random selection of functions, which leads to a stochastic average gradient (SAG) method, while the studies in [13], [16], and [17] use a cyclic scheme.

Deterministic quasi-Newton methods (where we have access to the exact gradient) have been studied extensively over the last 50 years. The main promise of quasi-Newton methods is to approximate the curvature of the function without computing the Hessian or its inverse, and indeed, there are several different approaches to accomplish this goal leading to different algorithms, but, perhaps, the most popular quasi-Newton algorithms are the symmetric rank-one (SR1) method [24], Broyden's method [25]–[27], the Davidon–Fletcher–Powell (DFP) method [28], [29], the BFGS method [1]–[4], and the limited-memory BFGS (L-BFGS) method [30], [31]. As mentioned earlier, the main advantage of quasi-Newton methods is that, while they only use the first-order information in their update, most of them converge to the optimal solution at a superlinear rate. This result has been established for a large group of quasi-Newton algorithms, including Broyden's method [1], [26], [32], the DFP method [26], [33], [34], the BFGS method [26], [34]–[36], and several other variants of these methods [37]–[43]. For introduction into the topic, refer [44, Ch. 6].

Stochastic quasi-Newton methods appear first in [5] and [7], and their convergence is first proved in [6] and [8]. An alternative provably convergent stochastic quasi-Newton method is proposed in [45]. It differs from those in [6]–[8] in which it collects the (stochastic) second-order information to estimate the objective's curvature. This is in contrast to estimating curvature using the difference of two consecutive stochastic gradients; we briefly discuss this method in Section V-A. Combining the ideas of randomness reduction and quasi-Newton methods is first proposed in [18] and [46], which leads to linearly convergent methods. Another linearly convergent stochastic quasi-Newton method is proposed in [47] in which the estimate of the inverse Hessian matrix is updated at each iteration using a sketch of the Hessian. Generalizing the theory of stochastic quasi-Newton methods to the nonstrong convex setting is studied in [48] and the nonsmooth convex setting is shown in [49]. Distributed extension of stochastic quasi-Newton is also studied in [50] and [51]. Also, several studies analyze stochastic quasi-Newton methods for nonconvex losses [52]–[54].

## II. DETERMINISTIC QUASI-NEWTON OPTIMIZATION

For the function $f(\mathbf{w})$ in (2), we denote the gradient at $\mathbf{w}$ as $\mathbf{s}(\mathbf{w}) := \nabla f(\mathbf{w})$. Although exact computation of the gradient is impractical (see Section III), let us set aside this concern for a moment and use $\mathbf{s}(\mathbf{w})$ to implement a descent method. We do so with the introduction of a discrete index $t$, step sizes $\eta_t$, and positive-definite (PD) matrices $\mathbf{B}_t$ to define the iteration

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \, \mathbf{B}_t^{-1} \mathbf{s}(\mathbf{w}_t). \tag{3}$$

If the Hessian $\mathbf{H}(\mathbf{w}) := \nabla^2 f(\mathbf{w})$ has eigenvalues uniformly bounded away from 0 and uniformly bounded from above, then the variable iterates $\mathbf{w}_t$ converge to the optimal argument $\mathbf{w}^*$ if the step size sequence $\eta_t$ is properly chosen, and the matrices $\mathbf{B}_t$ also have eigenvalues uniformly bounded away from 0 and uniformly bounded from above.

Convergence holds irrespective of the specific choice of the $\mathbf{B}_t$ matrices, but different selections have dramatic effects in convergence rates. For instance, we can experience the linear convergence rate of gradient descent if we choose identity matrices by setting $\mathbf{B}_t = \mathbf{I}$, or we can experience the (local) quadratic convergence rate of Newton's method if we choose $\mathbf{B}_t = \mathbf{H}(\mathbf{w}_t)$. The idea of quasi-Newton methods is to use matrices $\mathbf{B}_t$ that attain superlinear convergence toward the optimal argument $\mathbf{w}^*$. This is characterized by a sequence of iterates that satisfy the limit property

$$\limsup_{t \to \infty} \frac{\|\mathbf{w}_{t+1} - \mathbf{w}^*\|}{\|\mathbf{w}_t - \mathbf{w}^*\|} = 0. \tag{4}$$

This is not as good as the quadratic rate of Newton's method, but it is much better than the linear convergence rate of gradient descent—the limit in (4) would be a number in the interval $(0, 1)$.

To attain superlinear convergence, the quasi-Newton methods select matrices that satisfy the secant condition

$$\mathbf{B}_{t+1}(\mathbf{w}_{t+1} - \mathbf{w}_t) = \mathbf{s}(\mathbf{w}_{t+1}) - \mathbf{s}(\mathbf{w}_t). \tag{5}$$

Observe that, per (5), the matrix $\mathbf{B}_{t+1}$ satisfies the secant condition for the pair of iterates $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$, but, consistent with (3), this matrix is used to generate the iterate

$\mathbf{w}_{t+2}$. To streamline upcoming discussions, define the variable and gradient variations at time $t$ as

$$\mathbf{v}_t := \mathbf{w}_{t+1} - \mathbf{w}_t, \quad \text{and } \mathbf{r}_t := \mathbf{s}(\mathbf{w}_{t+1}) - \mathbf{s}(\mathbf{w}_t) \tag{6}$$

so that (5) can be more succinctly written as $\mathbf{B}_{t+1}\mathbf{v}_t = \mathbf{r}_t$.

A simple rationale for the secant condition in (5) is that the Hessian $\mathbf{H}(\mathbf{w}_t)$ satisfies this condition for vanishing distance $\|\mathbf{w}_{t+1} - \mathbf{w}_t\|$. We can, therefore, think of (5) as an attempt to choose a matrix that would render (3) close to Newton's method. A perhaps more insightful rationale is discussed in Remark 2, but, regardless of interpretation, it is important to note that the specification of $\mathbf{B}_{t+1}$ in (5) is indeterminate. To resolve this indeterminacy, the quasi-Newton methods require the matrix $\mathbf{B}_{t+1}$ to be close to the matrix $\mathbf{B}_t$. Exactly what close the entails varies for different quasi-Newton methods, but, over the years, the use of the BFGS method has become standard. In BGFS, the matrix $\mathbf{B}_{t+1}$ is chosen as the one that satisfies the secant condition while been closest to $\mathbf{B}_t$ in terms of the Gaussian differential entropy

$$\begin{aligned} \mathbf{B}_{t+1} \;=\; &\underset{\mathbf{Z}}{\text{argmin}} \;\; \text{tr}\big[\mathbf{B}_t^{-1}\mathbf{Z}\big] - \log\det\big[\mathbf{B}_t^{-1}\mathbf{Z}\big] - p \\ &\text{s.t.} \quad \mathbf{Z}\mathbf{v}_t = \mathbf{r}_t, \quad \mathbf{Z} \succeq \mathbf{0}. \end{aligned} \tag{7}$$

The problem in (7) is a convex semidefinite program that can be solved in the closed form [44]. At this point, however, it is more important to remark that the condition $\mathbf{Z} \succeq \mathbf{0}$ does not bound the eigenvalues of $\mathbf{B}_{t+1}$ away from zero. It is ready to see that, because of the log determinant function in the objective of (7), we will have $\mathbf{B}_{t+1} \succ \mathbf{0}$ if $\mathbf{B}_t \succ \mathbf{0}$. In fact, for strongly convex functions, it can be verified that the eigenvalues of $\mathbf{B}_t$ are larger than a positive constant [44, Ch. 6].

However, in general settings, it is not impossible for the smallest eigenvalue of $\mathbf{B}_t$ to approach 0 as time progresses. A simple solution to avoid this issue is to require a lower bound on the smallest eigenvalue of $\mathbf{B}_{t+1}$ by replacing (7) with

$$\begin{aligned} \mathbf{B}_{t+1} = &\underset{\mathbf{Z}}{\text{argmin}} \;\; \text{tr}\big[\mathbf{B}_t^{-1}(\mathbf{Z} - \delta\mathbf{I})\big] - \log\det\big[\mathbf{B}_t^{-1}(\mathbf{Z} - \delta\mathbf{I})\big] \\ &\text{s.t.} \quad \mathbf{Z}\mathbf{v}_t = \mathbf{r}_t, \quad \mathbf{Z} \succeq \mathbf{0}. \end{aligned} \tag{8}$$

Using the sequence of matrices defined by the proximity condition in (8) yields the regularized BFGS method [6]. It is a method that still abides to the secant condition in (5) but comes with a guarantee that the smallest eigenvalue of $\mathbf{B}_t$ is at least $\delta > 0$ for all iterations.

As is the case with (7), the semidefinite program in (8) also has an analytic solution. To write it down, we define the corrected gradient variation as

$$\tilde{\mathbf{r}}_t := \mathbf{r}_t - \delta\mathbf{v}_t \tag{9}$$

and use it to write the solution of (8) as

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\tilde{\mathbf{r}}_t\tilde{\mathbf{r}}_t^T}{\mathbf{v}_t^T\tilde{\mathbf{r}}_t} - \frac{\mathbf{B}_t\mathbf{v}_t\mathbf{v}_t^T\mathbf{B}_t}{\mathbf{v}_t^T\mathbf{B}_t\mathbf{v}_t} + \delta\mathbf{I}. \tag{10}$$

Incidentally, this expression is also a solution to (8) if we make $\delta = 0$ in (9) and (10). An interesting observation about the regularized BFGS matrix in (10) is that it is the term $\delta\mathbf{I}$ that makes the smallest eigenvalue of $\mathbf{B}_{t+1}$ larger than $\delta$. However, this regularization is not equivalent to just adding a $\delta\mathbf{I}$ term to the solution of (8). Doing so would guarantee that the smallest eigenvalue of the resulting matrix is at least $\delta$ but would yield a matrix that violates the secant condition. To maintain validity of the secant condition, we need to modify the gradient variation $\mathbf{r}_t$ of (6) as specified in (9). The (regularized) BFGS method is defined by the descent iteration in (3) where the curvature estimation matrix $\mathbf{B}_t$ is as given by the recursion in (10). The stochastic quasi-Newton methods replace the gradients that appear in the descent iteration and those that appear in the curvature estimation in (10) with stochastic gradients. We explain this in Section III after a few remarks.

*Remark 1:* A solution to the problem in (8) need not exist, that is, the problem may be infeasible. A sufficient condition for a solution to exist is for the inner product of the modified gradient variation $\tilde{\mathbf{r}}_t$ and the variable variation $\mathbf{v}_t$ to satisfy $\tilde{\mathbf{r}}_t^T\mathbf{v}_t = (\mathbf{r}_t - \delta\mathbf{v}_t)^T\mathbf{v}_t \geq 0$ [6, Proposition 1]. In turn, this is guaranteed to hold for all variable pairs $(\mathbf{w}_t, \mathbf{w}_{t+1})$ if the smallest eigenvalue of the Hessians of $f$ is larger than or equal to $\delta$ [6, Lemma 1]. This is an intuitive condition because requiring $\mathbf{B}_{t+1} \succeq \delta\mathbf{I}$ means that it is impossible to satisfy secant conditions where the curvature of $f$ is shallower than $\delta$.

*Remark 2:* To gain further insights into the value of the secant condition in (5), set $\eta_t = 1$ in (3) and reorder the terms to write $\mathbf{w}_{t+1} - \mathbf{w}_t = -\mathbf{B}_t^{-1}\mathbf{s}(\mathbf{w}_t)$. Substituting this expression into the secant condition and reordering terms lead to

$$\mathbf{s}(\mathbf{w}_{t+1}) = \big(\mathbf{I} - \mathbf{B}_{t+1}\mathbf{B}_t^{-1}\big)\mathbf{s}(\mathbf{w}_t). \tag{11}$$

If we had $\mathbf{B}_{t+1} = \mathbf{B}_t$, the term $\mathbf{B}_{t+1}\mathbf{B}_t^{-1}$ would cancel out the identity matrix $\mathbf{I}$, and we would have $\mathbf{s}(\mathbf{w}_{t+1}) = \mathbf{0}$, which would, in turn, imply that $\mathbf{w}_{t+1} = \mathbf{w}^*$. This does not happen exactly, but, as iterations grow, the secant conditions for the pair of iterates $(\mathbf{w}_t, \mathbf{w}_{t+1})$ and $(\mathbf{w}_{t-1}, \mathbf{w}_t)$ become similar, and the matrices $\mathbf{B}_t$ and $\mathbf{B}_{t+1}$ become close. According to (11), this closeness results in a rapid decrease of the gradient $\mathbf{w}_t$ and explains the superlinear convergence of BFGS.

## III. STOCHASTIC BFGS ALGORITHMS

Recall the definition of $f(\mathbf{w}) := (1/N)\sum_{i=1}^N f(\mathbf{w}, \boldsymbol{\theta}_i)$ and assume that all component functions $f(\mathbf{w}, \boldsymbol{\theta}_i)$ are differentiable. Hence, the computation of gradient $\mathbf{s}(\mathbf{w}) := \nabla f(\mathbf{w})$ is infeasible if the number of component functions $N$ is

large and motivates the use of stochastic gradients. These are defined as follows:

$$\hat{\mathbf{s}}(\mathbf{w},\tilde{\boldsymbol{\theta}}) := \frac{1}{b}\sum_{l=1}^{b}\nabla f(\mathbf{w},\boldsymbol{\theta}_l) \tag{12}$$

where $\tilde{\boldsymbol{\theta}} = [\boldsymbol{\theta}_1;\ldots;\boldsymbol{\theta}_b]$ is a set of $b \ll N$ realizations chosen independently and uniformly at random from the training set. Note that the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w},\tilde{\boldsymbol{\theta}})$ is an unbiased estimate of the gradient, namely $\mathbb{E}[\hat{\mathbf{s}}(\mathbf{w},\tilde{\boldsymbol{\theta}})] = \mathbf{s}(\mathbf{w})$.

Paralleling (3), we can build a stochastic descent method if we replace the gradient $\mathbf{s}(\mathbf{w}_t)$ with a stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$. Doing so results in the iteration

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t\,\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t) \tag{13}$$

where we replaced $\mathbf{B}_t$ with $\hat{\mathbf{B}}_t$ to emphasize that they are chosen according to different criteria. As is the case of (3), the SGD iterates generated by (13) can be shown to converge to $\mathbf{w}^*$ under the same eigenvalue restrictions for $\hat{\mathbf{B}}_t$ and the Hessians $\mathbf{H}(\mathbf{w})$, along with some further mild restrictions on the step sizes $\eta_t$ and the second moment of the stochastic gradient norm.

SGD is the method that results from making $\hat{\mathbf{B}}_t = \mathbf{I}$ in (13). SGD is the most used method to solve ERM problems. Its common use, notwithstanding, is that SGD inherits the slow converge of gradient descent. The stochastic quasi-Newton methods utilize different criteria to select matrices $\hat{\mathbf{B}}_t$ that will result in faster convergence. For instance, the online (o)BFGS [7] method is obtained if $\hat{\mathbf{B}}_t^{-1}$ satisfies a stochastic version of the secant condition in (5). That is, we seek a sequence of matrices such that

$$\hat{\mathbf{B}}_{t+1}(\mathbf{w}_{t+1}-\mathbf{w}_t) = \hat{\mathbf{s}}(\mathbf{w}_{t+1},\tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t). \tag{14}$$

As we have already seen in the case of deterministic quasi-Newton methods, the condition in (14) is underdetermined, but this is something we resolve with a proximity condition. If we adopt the proximity condition in (7), we know that (10) with $\delta = 0$ provides a closed-form expression for the evolution of $\hat{\mathbf{B}}_t$. We can, therefore, define oBFGS as a method, where $\mathbf{w}_t$ evolves per (13) with the matrix sequence $\hat{\mathbf{B}}_t$ evolving as

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t + \frac{\hat{\mathbf{r}}_t\hat{\mathbf{r}}_t^T}{\mathbf{v}_t^T\hat{\mathbf{r}}_t} - \frac{\hat{\mathbf{B}}_t\mathbf{v}_t\mathbf{v}_t^T\hat{\mathbf{B}}_t}{\mathbf{v}_t^T\hat{\mathbf{B}}_t\mathbf{v}_t} \tag{15}$$

where we have defined the stochastic gradient variation $\hat{\mathbf{r}}_t$ as

$$\hat{\mathbf{r}}_t := \hat{\mathbf{s}}(\mathbf{w}_{t+1},\tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t). \tag{16}$$

The oBFGS method is handicapped by the fact that the largest eigenvalue of $\hat{\mathbf{B}}_t$ is not guaranteed to stay bounded and the smallest eigenvalue of $\hat{\mathbf{B}}_t$ is not guaranteed to stay bounded away from 0. The RES algorithm introduces respective regularizations to overcome these problems [6]. To explicitly explain RES, begin by modifying the descent iteration in (13) by introducing a (small) constant $\Gamma$ and adding $\Gamma\mathbf{I}$ to $\hat{\mathbf{B}}_t^{-1}$ so that the iterates $\mathbf{w}_t$ are given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t\big(\hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I}\big)\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t). \tag{17}$$

If some eigenvalues of $\hat{\mathbf{B}}_t$ grow unbounded, progress along the associated eigenvector stalls because the corresponding eigenvalue of $\hat{\mathbf{B}}_t^{-1}$ approaches 0. The addition of the factor $\Gamma\mathbf{I}$ guarantees progress toward the minimum in all directions and, therefore, regularizes with respect to possible large eigenvalues of $\hat{\mathbf{B}}_t$. The more significant problem with oBFGS stems from eigenvalues of $\hat{\mathbf{B}}_t$ that may approach 0. As was the case in Section II, the matrix $\hat{\mathbf{B}}_t$ is guaranteed to stay PD, but there is no guarantee that its smallest eigenvalue stays bounded away from 0. In a stochastic optimization method, this is a fatal flaw. One can think of the stochastic gradients $\hat{\mathbf{s}}(\mathbf{w},\tilde{\boldsymbol{\theta}})$ of (12) as noisy versions of the true gradients $\mathbf{s}(\mathbf{w})$. An eigenvalue of $\hat{\mathbf{B}}_t$ close to 0 is a very large eigenvalue in the inverse matrix $\hat{\mathbf{B}}_t^{-1}$ in (13). This large eigenvalue amplifies the stochastic gradient noise drawing $\mathbf{w}_t$ away from convergence.

The resolution to this problem is the use of the matrices that follow from the regularized proximity condition in (8). Then, redefine the modified variation $\tilde{\mathbf{r}}_t$ in (10) so that it stands for the modified stochastic gradient variation

$$\tilde{\mathbf{r}}_t := \hat{\mathbf{r}}_t - \delta\mathbf{v}_t. \tag{18}$$

We further redefine the Hessian approximation $\hat{\mathbf{B}}_{t+1}$ for the next iteration as the matrix that satisfies the stochastic secant condition in (14) but is closest to $\hat{\mathbf{B}}_t$ in the sense of (8)—as opposed to being closest in the sense of (7). Per (10), we can compute $\hat{\mathbf{B}}_{t+1}$ explicitly as

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t + \frac{\tilde{\mathbf{r}}_t\tilde{\mathbf{r}}_t^T}{\mathbf{v}_t^T\tilde{\mathbf{r}}_t} - \frac{\hat{\mathbf{B}}_t\mathbf{v}_t\mathbf{v}_t^T\hat{\mathbf{B}}_t}{\mathbf{v}_t^T\hat{\mathbf{B}}_t\mathbf{v}_t} + \delta\mathbf{I}. \tag{19}$$

By construction, the matrices $\hat{\mathbf{B}}_t$ generated through (19) have eigenvalues that are not smaller than $\delta$. This implies that the stochastic gradient noise is controlled at all iterations because no eigenvalue of $\hat{\mathbf{B}}_t^{-1}$ can exceed $1/\delta$. This regularization is fundamental in avoiding catastrophic noise amplification.

A block diagram for RES is shown in Fig. 1. The inputs to each iteration are the current variable iterate $\mathbf{w}_t$, the current Hessian approximation $\hat{\mathbf{B}}_t$, and a random selection of realizations $\tilde{\boldsymbol{\theta}}_t$. The variable $\mathbf{w}_t$ and the random selection of realizations $\tilde{\boldsymbol{\theta}}_t$ generate the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$, which is fed along with the variable $\mathbf{w}_t$ and the matrix $\hat{\mathbf{B}}_t$ into a block that implements (17) to produce the next variable iterate $\mathbf{w}_{t+1}$. For the variable $\mathbf{w}_{t+1}$, we compute the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_{t+1},\tilde{\boldsymbol{\theta}}_t)$ corresponding to the same choice of functions $\tilde{\boldsymbol{\theta}}_t$ used to compute the

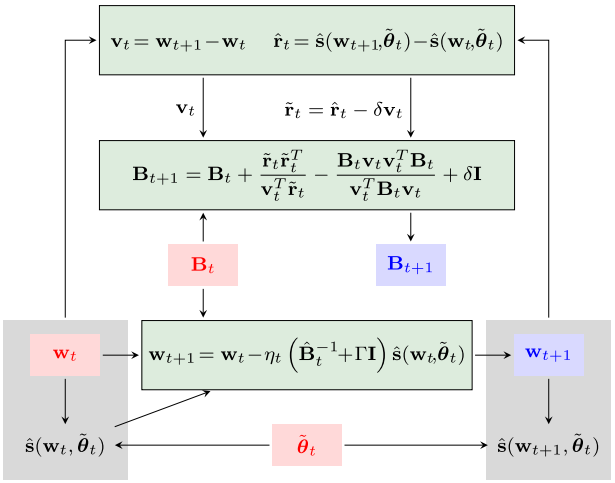



**Fig. 1.** *Regularized stochastic BFGS.*

stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$. The variable pair $(\mathbf{w}_t, \mathbf{w}_{t+1})$ and the stochastic gradient pair $(\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t), \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t))$ feed a block whose outputs are the variable variation $\mathbf{v}_t := \mathbf{w}_{t+1} - \mathbf{w}_t$ [see (6)] and the stochastic gradient variation in (16). These variations are used to update the curvature approximation per (19). A block diagram for oBFGS is obtained if we make $\delta = 0$ and $\Gamma = 0$ in Fig. 1.

*Remark 3:* One may think that the natural substitution of the gradient variation $\mathbf{r}_t = \mathbf{s}(\mathbf{w}_{t+1}) - \mathbf{s}(\mathbf{w}_t)$ is the stochastic gradient variation $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_{t+1}) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ instead of the variation $\hat{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ in (16). This would have the advantage that $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_{t+1})$ is the stochastic gradient used to descend in iteration $t+1$, whereas $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t)$ is not and is just computed for the purposes of updating $\mathbf{B}_t$. Therefore, using the variation $\hat{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ requires twice as many stochastic gradient evaluations as using the variation $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_{t+1}) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$. However, the use of the variation $\hat{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ is necessary to ensure that $(\hat{\mathbf{r}}_t - \delta \mathbf{v}_t)^T \mathbf{v}_t = \tilde{\mathbf{r}}_t^T \mathbf{v}_t > 0$, which, in turn, is required for (15) and (19) to be true. This cannot be guaranteed if we use the variation $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_{t+1}) - \hat{\mathbf{s}}(\mathbf{w}_t \tilde{\boldsymbol{\theta}}_t)$.

*Remark 4:* RES achieves a convergence rate of $\mathcal{O}(1/t)$ in expectation, when the objective function is strongly convexity and smooth, and the second moment of the stochastic gradient is bounded [6]. It can be shown that the same convergence rate can also be achieved when the last condition on the second moment of the stochastic gradient norm is relaxed to assuming that the variance of stochastic gradients is bounded. For more details regarding the convergence of RES, refer [6].

## IV. ONLINE LIMITED MEMORY BFGS ALGORITHM

The stochastic BFGS methods introduced in Section III speed up the convergence because they provide a means for estimating the curvature of a function in stochastic settings. It must be noted that this comes at the cost of increased computational cost per iteration. Instead of incurring the $\mathcal{O}(p)$ cost of SGD, the BFGS methods incur the $\mathcal{O}(p^2)$ cost of multiplying stochastic gradients with the matrix $\hat{\mathbf{B}}_t^{-1}$ [see (13)] and the $\mathcal{O}(p^3)$ cost of computing this inverse. The latter is incurred because the updates in (15) and (19) are for the matrices $\hat{\mathbf{B}}_t$, but we need their inverses $\hat{\mathbf{B}}_t^{-1}$ in (13). Whether this increased cost per iteration is justified depends on the balance between $p$ and the curvature of the function $f(\mathbf{w})$. In general, as the dimension of the variable $\mathbf{w} \in \mathbb{R}^p$ grows, oBFGS and RES become less appealing. Limited memory methods have been developed in deterministic [30], [31] and stochastic [7], [8] settings to extend the reach of quasi-Newton methods to problems with large dimension $p$. This section describes the online (o) limited memory (L) BFGS algorithm.

### A. Alternative Form for the oBFGS Method

Before we introduce oLBFGS, consider the update of the oBFGS curvature approximation matrix in (14). We see that the matrix $\hat{\mathbf{B}}_{t+1}$ is derived from the matrix $\hat{\mathbf{B}}_t$ though a pair of rank-1 updates. Saved for constant scalar factors, the first update is determined by the outer product $\hat{\mathbf{r}}_t \hat{\mathbf{r}}_t^T$ of the variable variation with itself, and the second rank-1 update is characterized by the outer product $(\hat{\mathbf{B}}_t \mathbf{v}_t)(\hat{\mathbf{B}}_t \mathbf{v}_t)^T$ of the vector $\hat{\mathbf{B}}_t \mathbf{v}_t$ with itself. Since this is true, we can use the Sherman–Morrison formula to relate the inverse $\hat{\mathbf{B}}_{t+1}^{-1}$ with the inverse $\hat{\mathbf{B}}_t^{-1}$ without going through the intermediate computation of the matrix $\hat{\mathbf{B}}_{t+1}$. The implementation of these computations requires some cumbersome algebra, but the final result is that we can compute $\hat{\mathbf{B}}_{t+1}^{-1}$ from $\hat{\mathbf{B}}_t^{-1}$ using the update

$$\hat{\mathbf{B}}_{t+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{v}_t \hat{\mathbf{r}}_t^T}{\hat{\mathbf{r}}_t^T \mathbf{v}_t}\right) \hat{\mathbf{B}}_t^{-1} \left(\mathbf{I} - \frac{\hat{\mathbf{r}}_t \mathbf{v}_t^T}{\hat{\mathbf{r}}_t^T \mathbf{v}_t}\right) + \frac{\mathbf{v}_t \mathbf{v}_t^T}{\hat{\mathbf{r}}_t^T \mathbf{v}_t}. \tag{20}$$

In Section III, oBFGS is the method defined by the recursion in (13) with matrices $\hat{\mathbf{B}}_t$ evolving as determined by (15). In light of (20), oBFGS is equivalent to recursive application of (13) with matrices $\hat{\mathbf{B}}_t^{-1}$ evolving per (20). This alternative definition bypasses computation and inversion of $\hat{\mathbf{B}}_t$. Doing so, the computational cost per iteration is reduced to $\mathcal{O}(p^2)$ because all the matrices involved other than $\hat{\mathbf{B}}_t^{-1}$ are rank-1 matrices. It is important to remark that an analogous reduction in computational cost is not known to hold for RES.

oBFGS can be implemented with a computational cost of order $\mathcal{O}(p^2)$, which itself is interesting. A perhaps more interesting conclusion can be drawn if we consider the first iteration of (20) assuming that the initial condition is the identity matrix. To make matters clear, make $t = 0$ in (20) and initialize the curvature approximation matrix to $\hat{\mathbf{B}}_0^{-1} = \mathbf{I}$. Doing so results in the matrix $\hat{\mathbf{B}}_1^{-1}$ taking the form

$$\hat{\mathbf{B}}_1^{-1} = \left(\mathbf{I} - \frac{\mathbf{v}_0 \hat{\mathbf{r}}_0^T}{\hat{\mathbf{r}}_0^T \mathbf{v}_0}\right) \left(\mathbf{I} - \frac{\hat{\mathbf{r}}_0 \mathbf{v}_0^T}{\hat{\mathbf{r}}_0^T \mathbf{v}_0}\right) + \frac{\mathbf{v}_0 \mathbf{v}_0^T}{\hat{\mathbf{r}}_0^T \mathbf{v}_0}. \tag{21}$$

The important point to make about (21) is that $\hat{\mathbf{B}}_1^{-1}$ is given by a sum of outer products. Thus, if we want to compute the product of $\hat{\mathbf{B}}_1^{-1}\hat{\mathbf{s}}(\mathbf{w}_1,\tilde{\boldsymbol{\theta}}_1)$ in order to implement (13), the computational cost is of order $\mathcal{O}(p)$, not $\mathcal{O}(p^2)$. If we now consider the second iteration with $t=1$, we have that $\hat{\mathbf{B}}_1^{-1}$ is a sum of outer products and that we, therefore, are able to compute $\hat{\mathbf{B}}_2^{-1}$ as per (20) and still end up with a sum of outer products. Thus, to compute the inner product $\hat{\mathbf{B}}_2^{-1}\hat{\mathbf{s}}(\mathbf{w}_2,\tilde{\boldsymbol{\theta}}_2)$, the computational cost is still of $\mathcal{O}(p)$, not $\mathcal{O}(p^2)$. Arguing recursively, this can be proven true for any time index $t$. Of course, as the iteration index grows, the number of outer product summands that define $\hat{\mathbf{B}}_t^{-1}$ also grows. At some point, it is cheaper to use (20) directly and incur the $\mathcal{O}(p^2)$ cost of computing the product $\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$ than it is to compute the inner product of each separate summand with $\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$. As we will explain in Section IV-C, the number of summands that define $\hat{\mathbf{B}}_t^{-1}$ grows linearly with the time iteration index $t$. Thus, when $t \ll p$, it is computationally more efficient to evaluate $\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$ using the fact that $\hat{\mathbf{B}}_t^{-1}$ is a sum of outer products than it is to use (20) directly and compute the product $\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$. Since the computational advantage of writing $\hat{\mathbf{B}}_t^{-1}$ as a sum of outer products disappears and the iteration index grows, the idea of the oLBFGS method is to reset the curvature approximation matrix at all iterations. We explain this in Section IV-B.

## B. oLBFGS Algorithm

To simplify the notation, introduce the scalar $\rho_t$ and the matrix $\mathbf{Z}_t$ given by the expressions

$$\rho_t := 1/(\hat{\mathbf{r}}_t^T\mathbf{v}_t), \quad \mathbf{Z}_t := \mathbf{I} - \rho_t\hat{\mathbf{r}}_t\mathbf{v}_t^T. \tag{22}$$

Using these definitions, the oBFGS update for the matrix $\hat{\mathbf{B}}_{t+1}^{-1}$ in (20) can be simplified to

$$\hat{\mathbf{B}}_{t+1}^{-1} = \mathbf{Z}_t^T\,\hat{\mathbf{B}}_t^{-1}\,\mathbf{Z}_t + \rho_t\,\mathbf{v}_t\mathbf{v}_t^T. \tag{23}$$

oLBFGS uses (23) not as a recursion to produce the sequence of matrices $\hat{\mathbf{B}}_t^{-1}$ used in (13) but as an inner loop of $\tau$ recursive iterations that start from a given initialization to produce a matrix $\hat{\mathbf{B}}_t^{-1}$ to be used in (13). Suppose that the current outer iteration index is $t$, and consider a window made up of the last $\tau$ variable and stochastic gradient variations

$$\mathcal{V}_t = \{(\mathbf{v}_u,\hat{\mathbf{r}}_u), \quad \text{for } \tilde{t} := t-\tau \le u \le t-1\}. \tag{24}$$

The set $\mathcal{V}_t$ registers the history of past variable and stochastic gradient variations starting at $\tilde{t} := t-\tau$ and ending at $t-1$. Now, introduce an inner iteration index $u$ to write curvature approximation matrices of the form $\hat{\mathbf{B}}_{t,u}^{-1}$. Matrices $\hat{\mathbf{B}}_{t,u}^{-1}$ and $\hat{\mathbf{B}}_{t,u+1}^{-1}$ with subsequent inner indexes are related by [see (23)]

$$\hat{\mathbf{B}}_{t,u+1}^{-1} = \mathbf{Z}_{\tilde{t}+u}^T\,\hat{\mathbf{B}}_{t,u}^{-1}\,\mathbf{Z}_{\tilde{t}+u} + \rho_{\tilde{t}+u}\,\mathbf{v}_{\tilde{t}+u}\,\mathbf{v}_{\tilde{t}+u}^T. \tag{25}$$

The initial matrix $\hat{\mathbf{B}}_{t,0}^{-1}$ is given, and the time index is $u=0,\dots,\tau-1$. For inner iteration index $u=1$, the inverse curvature approximation matrix $\hat{\mathbf{B}}_{t,1}^{-1}$ updates $\hat{\mathbf{B}}_{t,0}^{-1}$ using the variable and stochastic gradient variations $(\mathbf{v}_{t-\tau},\hat{\mathbf{r}}_{t-\tau})$ corresponding to iteration $u=t-\tau:=\tilde{t}$. In the second iteration with $u=2$, the inverse curvature approximation matrix $\hat{\mathbf{B}}_{t,2}^{-1}$ updates $\hat{\mathbf{B}}_{t,1}^{-1}$ using the variable and stochastic gradient variations $(\mathbf{v}_{t-\tau+1},\hat{\mathbf{r}}_{t-\tau+1})$ corresponding to iteration $u=t-\tau+1=\tilde{t}+1$. As the iteration index $u$ advances, we use more recent elements of the set $\mathcal{V}_t$ defined in (24). The last update corresponds to iteration $u=t-1$ in which the matrix $\hat{\mathbf{B}}_{t,\tau}^{-1}$ is obtained from the matrix $\hat{\mathbf{B}}_{t,\tau-1}^{-1}$ using the variable and stochastic gradient variations $(\mathbf{v}_{t-1},\hat{\mathbf{r}}_{t-1})$ corresponding to iteration $u=t-1=\tilde{t}+\tau-1$. The oLBFGS algorithm is defined by the stochastic descent iteration in (26) in which we use this latter matrix as a curvature estimate. We do that by making $\hat{\mathbf{B}}_t^{-1}=\hat{\mathbf{B}}_{t,\tau}^{-1}$ in (13)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t\,\hat{\mathbf{B}}_{t,\tau}^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t) = \mathbf{w}_t - \eta_t\hat{\mathbf{d}}_t. \tag{26}$$

Observe that, in the second equality, we have defined the oLBFGS step $\hat{\mathbf{d}}_t := \hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t,\tilde{\boldsymbol{\theta}}_t)$ to simplify upcoming discussions. Notice that the inverse curvature approximation $\hat{\mathbf{B}}_t^{-1}$ in (25) is a function of the initial approximation $\mathbf{B}_{t,0}^{-1}$ and the $\tau$ most recent curvature information pairs recorded in the set $\mathcal{V}_t$ of (24). When $t<\tau$, there are not enough pairs $(\mathbf{v}_u,\hat{\mathbf{r}}_u)$ to perform $\tau$ updates. In such a case, we just redefine $\tau=t$ and proceed to use the $t=\tau$ available pairs $(\mathbf{v}_u,\hat{\mathbf{r}}_u)$ with $0\le u\le t-1$.

oLBFGS is defined by (26) with each of the matrices $\hat{\mathbf{B}}_{t,\tau}^{-1}$ computed by the $\tau$ inner loop iterations defined by (25). As we explained in Section IV-A, the implementation of the product $\mathbf{B}_t^{-1}\mathbf{s}(\mathbf{w}_t)$ in (3) for matrices $\mathbf{B}_t^{-1}=\mathbf{B}_{t,\tau}^{-1}$ obtained from the recursion in (25) does not need explicit computation of the matrix $\mathbf{B}_{t,\tau}^{-1}$. Rather, we can instead compute $\mathcal{O}(\tau)$ inner products (see Section IV-C for the details of this computation). Consequently, the implementation of the recursion in (25) does not need computation and storage of prior matrices $\mathbf{B}_{t-1}^{-1}$. Rather, it suffices to keep $\tau$ most recent curvature information pairs $(\mathbf{v}_u,\mathbf{r}_u)\in\mathcal{V}_t$, thus reducing storage requirements from $O(p^2)$ to $O(\tau p)$. Furthermore, each of these inner products can be computed at a cost of $p$ operations, yielding a total computational cost of $O(\tau p)$ per oLBFGS iteration. Hence, oLBFGS decreases both the memory requirements and the computational cost of each iteration from $O(p^2)$ required by regular oBFGS to $O(\tau p)$.

## C. Limiting the Computation Cost of oLBFGS

The equations in (25) and (26) are used conceptually but not in practical implementations. For the latter,

**Algorithm 1** Computation of oLBFGS Step $\mathbf{q} = \hat{\mathbf{B}}_t^{-1}\mathbf{p}$ When Called With $\mathbf{p} = \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$

1: **function** $\mathbf{q}$ = oLBFGS Step($\hat{\mathbf{B}}_{t,0}^{-1}, \mathbf{p} = \mathbf{p}_0, \{\mathbf{v}_u, \hat{\mathbf{r}}_u\}_{u=\tilde{t}}^{t-1}$)
2: **for** $u = 0, 1, \ldots, \tau - 1$ **do** [Loop to compute $\alpha_u$ and $\mathbf{p}_u$]
3: Compute and store scalar $\alpha_u = \rho_{t-u-1}\mathbf{v}_{t-u-1}^T\mathbf{p}_u$
4: Compute $\mathbf{p}_{u+1} = \mathbf{p}_u - \alpha_u\hat{\mathbf{r}}_{t-u-1}$.
5: **end for**
6: Multiply $\mathbf{p}_\tau$ by initial matrix: $\mathbf{q}_0 = \hat{\mathbf{B}}_{t,0}^{-1}\mathbf{p}_\tau$
7: **for** $u = 0, 1, \ldots, \tau - 1$ **do** [Loop to compute $\beta_u$ and $\mathbf{q}_u$]
8: Compute scalar $\beta_u = \rho_{\tilde{t}+u}\hat{\mathbf{r}}_{\tilde{t}+u}^T\mathbf{q}_u$
9: Compute $\mathbf{q}_{u+1} = \mathbf{q}_u + (\alpha_{\tau-u-1} - \beta_u)\mathbf{v}_{\tilde{t}+u}$
10: **end for** {return $\mathbf{q} = \mathbf{q}_\tau$}

we exploit the structure of (25) to rearrange the terms in the computation of the product $\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$. To see how this is done, consider the recursive update for the Hessian inverse approximation $\hat{\mathbf{B}}_t^{-1}$ in (25) and make $u = \tau - 1$ to write

$$\hat{\mathbf{B}}_t^{-1} = \hat{\mathbf{B}}_{t,\tau}^{-1} = \mathbf{Z}_{t-1}^T\hat{\mathbf{B}}_{t,\tau-1}^{-1}\mathbf{Z}_{t-1} + \rho_{t-1}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^T. \tag{27}$$

Equation (27) shows the relation between the Hessian inverse approximation $\hat{\mathbf{B}}_t^{-1}$ and the $(\tau - 1)$th updated version of the initial Hessian inverse approximation $\hat{\mathbf{B}}_{t,\tau-1}^{-1}$ at step $t$. We can proceed, recursively, to show that $\hat{\mathbf{B}}_t^{-1}$ can be written as

$$\begin{aligned}\hat{\mathbf{B}}_t^{-1} = {} & \big(\mathbf{Z}_{t-1}^T \ldots \mathbf{Z}_{\tilde{t}}^T\big)\hat{\mathbf{B}}_{t,0}^{-1}(\mathbf{Z}_{\tilde{t}} \ldots \mathbf{Z}_{t-1}) \\ & + \hat{\rho}_{\tilde{t}}\big(\mathbf{Z}_{t-1}^T \ldots \mathbf{Z}_{\tilde{t}+1}^T\big)\mathbf{v}_{\tilde{t}}\mathbf{v}_{\tilde{t}}^T(\mathbf{Z}_{\tilde{t}+1} \ldots \mathbf{Z}_{t-1}) \\ & + \cdots + \rho_{t-2}\big(\mathbf{Z}_{t-1}^T\big)\mathbf{v}_{t-2}\mathbf{v}_{t-2}^T(\mathbf{Z}_{t-1}) \\ & + \rho_{t-1}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^T.\end{aligned} \tag{28}$$

Note that the matrix $\mathbf{Z}_{t-1}$ and its transpose $\mathbf{Z}_{t-1}^T$ are the first and last product terms of all summands except the last, the matrix $\mathbf{Z}_{t-2}$ and its transpose $\mathbf{Z}_{t-2}^T$ are second and penultimate in all terms but the last two, and so on. Thus, when computing the oLBFGS step $\hat{\mathbf{d}}_t := \hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$, the operations needed to compute the product with the next to last summand of (28) can be reused to compute the product with the second to last summand, which, in turn, can be reused in determining the product with the third to last summand and so on. This observation compounded with the fact that multiplications with the identity plus rank 1 matrices $\mathbf{Z}_{t-1}$ require $O(p)$ operations yields an algorithm that can compute the oLBFGS step $\hat{\mathbf{d}}_t := \hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ in $O(\tau p)$ operations.

The computation of the product $\hat{\mathbf{B}}_t^{-1}\mathbf{p}$ is summarized in algorithmic form in the function in Algorithm 1. The function receives as arguments the initial matrix $\hat{\mathbf{B}}_{t,0}^{-1}$, the sequence of variable and stochastic gradient variations $(\mathbf{v}_u, \mathbf{r}_u) \in \mathcal{V}_t$, and the vector $\mathbf{p}$ to produce the outcome $\mathbf{q} = \mathbf{q}_\tau = \hat{\mathbf{B}}_t^{-1}\mathbf{p}$. When called with the stochastic gradient $\mathbf{p} = \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$, the function outputs the oLBFGS step $\hat{\mathbf{d}}_t := \hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ needed to implement the oLBFGS descent step in (26). The core of Algorithm 1 is given by the loop in steps 2–5 that compute the constants $\alpha_u$ and sequence elements $\mathbf{p}_u$ and the loop in steps 7–10 that compute the constants $\beta_u$ and sequence elements $\mathbf{q}_u$. The two loops are linked by the initialization of the second sequence with the outcome of the first, which is performed in step 6. To implement the first loop, we require $\tau$ inner products in step 4 and $\tau$ vector summations in step 5, which yields a total of $2\tau p$ multiplications. Likewise, the second loop requires $\tau$ inner products and $\tau$ vector summations in steps 9 and 10, respectively, which yields a total cost of also $2\tau p$ multiplications. Since the initial Hessian inverse approximation matrix $\hat{\mathbf{B}}_{t,0}^{-1}$ is diagonal, the cost of computation $\hat{\mathbf{B}}_{t,0}^{-1}\mathbf{p}_\tau$ in step 6 is $p$ multiplications. Thus, Algorithm 1 requires a total of $(4\tau + 1)p$ multiplications, which affirms the complexity cost of order $O(\tau p)$ for oLBFGS.

*Remark 5:* Under the same conditions required for RES, as mentioned in Remark 5, the oLBFGS method converges to the optimal solution at a rate of $\mathcal{O}(1/t)$ in expectation [8], while its computation cost per iteration is $\mathcal{O}(\tau p)$ instead of the $\mathcal{O}(p^3)$ computation cost per iteration of RES. For more details regarding the convergence properties of oLBFGS, refer [8].

**Algorithm 2** oLBFGS

**Require:** Initial iterate $\mathbf{w}_0$ and Hessian approx. parameter $\hat{\gamma}_0 = 1$.
1: **for** $t = 0, 1, 2, \ldots$ **do**
2: Acquire $b$ independent samples $\tilde{\boldsymbol{\theta}}_t = [\boldsymbol{\theta}_{t1}, \ldots, \boldsymbol{\theta}_{tb}]$
3: Compute $\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t) = \dfrac{1}{b}\displaystyle\sum_{l=1}^{b}\nabla_{\mathbf{w}} f(\mathbf{w}_t, \boldsymbol{\theta}_{tl})$
4: Find $\hat{\mathbf{B}}_{t,0}^{-1} = \hat{\gamma}_t\mathbf{I}$ with $\hat{\gamma}_t = \dfrac{\mathbf{v}_{t-1}^T\hat{\mathbf{r}}_{t-1}}{\hat{\mathbf{r}}_{t-1}^T\hat{\mathbf{r}}_{t-1}}$ for $t > 0$
5: Compute the descent direction $\hat{\mathbf{d}}_t$ with Algorithm 1: $\hat{\mathbf{d}}_t = \text{oLBFGS Step}\left(\hat{\mathbf{B}}_{t,0}^{-1},\ \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t),\ \{\mathbf{v}_u, \hat{\mathbf{r}}_u\}_{u=\tilde{t}}^{t-1}\right)$
6: Descend along direction $\hat{\mathbf{d}}_t$: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t\hat{\mathbf{d}}_t$
7: Compute $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t) = \dfrac{1}{b}\displaystyle\sum_{l=1}^{b}\nabla_{\mathbf{w}} f(\mathbf{w}_{t+1}, \boldsymbol{\theta}_{tl})$
8: Update $\mathbf{v}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$, $\hat{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$
9: **end for**

## V. LINEARLY CONVERGENT SLBFGS METHOD

In Sections III and IV, we discussed RES and oLBFGS as two provably convergent stochastic quasi-Newton methods that accelerate the convergence of the stochastic first-order methods by properly approximating the objective function curvature. Although these methods are successful in expanding the application of quasi-Newton methods to

stochastic settings, their convergence rate is sublinear. This is not better than the convergence rate of SGD and, as is also the case in SGD, is a consequence of the stochastic approximation noise that necessitates the use of diminishing stepsizes. The stochastic quasi-Newton methods in [18] and [46] resolve this issue and achieve a linear convergence rate by using the variance reduction technique proposed in [12]. The fundamental idea of [12] is to reduce the noise of stochastic gradient approximation by computing the exact gradient in an outer loop to use it in an inner loop for gradient approximation. In this section, we focus on the variance-reduced stochastic variant of LBFGS proposed in [18]. This method can be considered as a combination of the stochastic quasi-Newton method proposed in [45] and the variance reduction technique studied in [12]. Next, we briefly mention the main ideas of [12] and [45] and then present the variance-reduced SLBFGS algorithm proposed in [18].

### A. Stochastic LBFGS With Access to Hessian-Vector Products

In the classic BFGS method, the gradient variation $\mathbf{r}_t := \mathbf{s}(\mathbf{w}_{t+1}) - \mathbf{s}(\mathbf{w}_t)$, required for the update of Hessian approximation, is computed by subtracting two consecutive gradients. Hence, a natural choice for the stochastic setting, as mentioned in Sections III and IV for RES and oLBFGS, respectively, is to replace the gradients with stochastic gradients evaluated with respect to the same set of random variables (see Remark 3). However, if the size of the minibatch used for the stochastic gradient evaluation is small, and as a result, the gradient estimates are very noisy, the resulted Hessian approximation could be negatively affected. To resolve this issue, as suggested in [45], one can use different batch sizes for gradient estimation required for the descent direction and the gradient variation required for the update of Hessian approximation to decouple the stochastic gradient and curvature estimate calculations. Since the batch size for curvature estimate calculations is larger than the stochastic gradient estimation for the update, it can be computed every $K$ iterations.

Especially, consider $t$ as time index of the iterates and $k$ as the time index for the iterates that we update the curvature estimate. In the stochastic quasi-Newton method proposed in [45], at each iteration $t$, we choose a batch of samples $\tilde{\boldsymbol{\theta}}_t = [\boldsymbol{\theta}_{t,1}, \ldots, \boldsymbol{\theta}_{t,b}]$ with size $b$ to compute the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$, and use this stochastic gradient to update the iterates according to the update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \, \hat{\mathbf{B}}_k^{-1} \hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t) \tag{29}$$

where $\hat{\mathbf{B}}_k^{-1}$ corresponds to the last Hessian approximation computed before the current iterate. Again, note that we use a different index for the Hessian approximation $\hat{\mathbf{B}}_k$ as it is updated every $K$ iterations, that is, it is updated when $\mod(t, K) = 0$.

At the iterations that we update the Hessian approximation, the main idea of the curvature update is similar to the one for oLBFGS except the fact that we use average iterates for computing variable variation and a larger batch of samples for computing gradient variation. To be more precise, if we define $\bar{\mathbf{w}}_k := \frac{1}{K}\sum_{i=t-K}^{t-1} \mathbf{w}_i$, as the average iterates from time $t-1$ to $t-k$, then the variable variation at time $k$ is given by

$$\mathbf{v}_k = \bar{\mathbf{w}}_k - \bar{\mathbf{w}}_{k-1}. \tag{30}$$

Furthermore, by approximating differences in gradients via the first-order Taylor's expansion, the approximate gradient variation can be computed via the following Hessian-vector product:

$$\hat{\mathbf{r}}_k := \nabla^2 \hat{F}(\mathbf{w}_k, \hat{\boldsymbol{\theta}}_k)\mathbf{v}_k \tag{31}$$

where $\nabla^2 \hat{F}(\mathbf{w}_k, \hat{\boldsymbol{\theta}}_k)$ is a subsampled Hessian defined as $\nabla^2 \hat{F}(\mathbf{w}_k, \hat{\boldsymbol{\theta}}_k) := (1/b_H)\sum_{\boldsymbol{\theta}\in\hat{\boldsymbol{\theta}}_k} \nabla^2 f(\mathbf{w}, \boldsymbol{\theta})$, where the set $\hat{\boldsymbol{\theta}}_k = [\hat{\boldsymbol{\theta}}_{k,1}, \ldots, \hat{\boldsymbol{\theta}}_{k,b_H}]$ has size $b_H$. Since the gradient variation $\hat{\mathbf{r}}_k$ and, consequently, the Hessian approximation $\hat{\mathbf{B}}_k$ are updated every $K$ iterations, we can afford to choose $b_H$ larger than the minibatch for stochastic gradient $b$. Note that performing the update in (31) requires access to the Hessian of $f(\mathbf{w}, \boldsymbol{\theta})$ and computing a Hessian-vector product every $K$ iterations, unlike RES and oLBFGS that are fully gradient-based. Note that when the variable variation $\mathbf{v}_k$ and gradient variation $\hat{\mathbf{r}}_k$ are computed, then the Hessian inverse approximation $\hat{\mathbf{B}}_k^{-1}$ is updated according the update of LBFGS.

### B. SVRG Variance Reduction Technique

Though stochastic methods often make rapid progress early on, the variance of the estimates of the gradient slows their convergence near the optimum. Convergence guarantees typically require diminishing step sizes. One promising line of work involves speeding up the convergence of stochastic first-order methods by reducing the variance of the gradient estimates. A popular approach for reducing the variance of the gradient estimates is the one proposed in [12], which leads to a linearly convergent stochastic variance-reduced gradient method, known as SVRG. The SVRG method succeeds in reducing the computational cost of deterministic first-order methods by computing a single gradient per iteration and using a delayed version of the average gradient to update the iterates.

The variance reduction scheme in SVRG can be decomposed into two steps: the outer loop and the inner loop. In the outer loop, which happens every $m$ iterations, we evaluate the full gradient at the current point, that is, if $\tilde{\mathbf{w}} = \mathbf{w}_t$ is the current iterate, then we compute the exact (full) gradient as

$$\nabla f(\tilde{\mathbf{w}}) = \frac{1}{N}\sum_{i=1}^{N} \nabla f(\tilde{\mathbf{w}}, \boldsymbol{\theta}_i). \tag{32}$$

Then, in the inner loop, we update the iterates according to

$$\hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t - \eta_t(\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t) - \hat{\mathbf{s}}(\tilde{\mathbf{w}}, \tilde{\boldsymbol{\theta}}_t) + \nabla f(\tilde{\mathbf{w}})) \tag{33}$$

where the stochastic gradients $\hat{\mathbf{s}}(\mathbf{w}_t, \tilde{\boldsymbol{\theta}}_t)$ and $\hat{\mathbf{s}}(\tilde{\mathbf{w}}, \tilde{\boldsymbol{\theta}}_t)$ are evaluated based on the same set of random set $\tilde{\boldsymbol{\theta}}_t$ but at two different points, that is, the current iterate $\mathbf{w}_t$ and the iterate that, last time, we computed the full gradient $\nabla f(\tilde{\mathbf{w}})$. Note that we only perform the update, in (32), every $m$ iterations where the optimal choice of this parameter is well studied in [12]. It can also be easily checked that the descent direction in (33) is an unbiased estimator of the exact gradient $\nabla f(\tilde{\mathbf{w}}_t)$ at the current iterate. More importantly, it can be shown that the variance of gradient estimation in (33) approaches 0 as the iterates approach the optimal solution [12].

### C. Linearly Convergent Stochastic LBFGS

Now, we are at the right point to introduce the linearly convergent SLBFGS proposed in [18], which is constructed based on two blocks: 1) the stochastic quasi-Newton scheme discussed in Section V-A and 2) the variance reduction technique studied in Section V-B. To be more specific, in SLBFGS, for estimating the gradient required for the descent direction, we use the variance reduction of SVRG defined in (33) and, for the Hessian approximation, we follow the update of LBFGS where the variable variation is the difference of averages iterates as defined in (30), and the gradient variation is computed according to the update in (31). The steps of SLBFGS are summarized in Algorithm 3. Steps 2–8 are similar to the SVRG update except step 8 in which we premultiply the gradient estimation $\mathbf{g}_k$ by the Hessian inverse approximation $\mathbf{B}_r^{-1}$. The operations in steps 9–15 correspond to the Hessian approximation update according to the quasi-Newton scheme discussed in Section V-A.

**Algorithm 3** SLBFGS

**Require:** Initial iterate $\mathbf{w}_0$ and Hessian approx. parameter $\hat{\gamma}_0 = 1$.

1: **for** $t = 0, 1, 2, \ldots$ **do**
2: Compute a full gradient $\mu_t := \nabla f(\tilde{\mathbf{w}}_t)$
3: Set $\mathbf{w}_0 = \tilde{\mathbf{w}}_t$
4: **for** $k = 0, 1, \ldots, m$ **do**
5: Acquire $\tilde{\boldsymbol{\theta}}_k = [\boldsymbol{\theta}_{k,1}, \ldots, \boldsymbol{\theta}_{k,b}]$
6: Compute stochastic gradients $\hat{\mathbf{s}}(\mathbf{w}_k, \tilde{\boldsymbol{\theta}}_k)$ and $\hat{\mathbf{s}}(\tilde{\mathbf{w}}_t, \tilde{\boldsymbol{\theta}}_k)$
7: Compute $\mathbf{g}_k := \hat{\mathbf{s}}(\mathbf{w}_k, \tilde{\boldsymbol{\theta}}_k) - \hat{\mathbf{s}}(\tilde{\mathbf{w}}_t, \tilde{\boldsymbol{\theta}}_k) + \mu_t$
8: Set $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{B}_r^{-1} \mathbf{g}_k$
9: **if** $\mod(k, K) = 0$ **then**
10: Set $r \leftarrow r + 1$
11: Set $\mathbf{u}_r = \frac{1}{K} \sum_{j=k-K}^{k-1} \mathbf{w}_j$
12: Acquire $\hat{\boldsymbol{\theta}}_r = [\boldsymbol{\theta}_{r,1}, \ldots, \boldsymbol{\theta}_{r,b_H}]$ for $\nabla^2 \hat{F}(\mathbf{u}_r, \hat{\boldsymbol{\theta}}_k)$
13: Compute $\mathbf{v}_r = \mathbf{u}_r - \mathbf{u}_{r-1}$ and $\hat{\mathbf{r}}_r = \nabla^2 \hat{F}(\mathbf{u}_r, \hat{\boldsymbol{\theta}}_k)\mathbf{v}_r$
14: Compute $\mathbf{B}_r^{-1}$ based on LBFGS
15: **end if**
16: **end for**
17: Set $\tilde{\mathbf{w}}_{t+1} = \mathbf{w}_m$
18: **end for**

## VI. SUPERLINEARLY CONVERGENT INCREMENTAL QUASI-NEWTON METHOD

At this point, we must remark on an interesting mismatch. The convergence rate of SGD is sublinear, and the convergence rate of deterministic GD is linear. The use of variance reduction techniques in SGD recovers the linear convergence rate of GD [12]. On the other hand, the convergence rate of the stochastic quasi-Newton methods is sublinear, and the convergence rate of the deterministic quasi-Newton methods is superlinear. The use of variance reduction in the stochastic quasi-Newton methods achieves linear convergence but does not recover a superlinear rate. Hence, a fundamental question remains unanswered: Is it possible to design an IQN method that recovers the superlinear convergence of deterministic quasi-Newton algorithms? Next, we show that the answer to this question is positive as the IQN method introduced in [19] achieves this goal. The IQN method is the first quasi-Newton method to achieve superlinear convergence while having a per iteration cost independent of the number of functions $N$—the cost per iteration is of order $\mathcal{O}(p^2)$.

### A. IQN: Incremental Aggregated BFGS

Next, we present the IQN method to solve the finite sum problem in (2). IQN is incremental in which, at each iteration, only the information associated with a single function $f_i$ is updated. The particular function is chosen by cyclically iterating through the $N$ functions. The IQN method is aggregated in which the aggregate of the most recently observed information of all $N$ functions is used to compute the updated variable $\mathbf{w}^{t+1}$.

In IQN, we consider $\mathbf{z}_1^t, \ldots, \mathbf{z}_N^t$ as the copies of the variable $\mathbf{w}$ at time $t$ associated with the functions $f_1, \ldots, f_N$, respectively. Likewise, define $\nabla f_i(\mathbf{z}_i^t)$ as the gradient corresponding to the $i$th function. Furthermore, consider $\mathbf{B}_i^t$ as a PD matrix that approximates the $i$th component Hessian $\nabla^2 f_i(\mathbf{w}^t)$. We refer to $\mathbf{z}_i^t$, $\nabla f_i(\mathbf{z}_i^t)$, and $\mathbf{B}_i^t$ as the information corresponding to the $i$th function $f_i$ at step $t$. Note that the functions' information is stored in a shared memory, as shown in Fig. 2. To introduce the IQN method, we first explain the mechanism for computing the updated variable $\mathbf{w}^{t+1}$ using the stored information $\{\mathbf{z}_i^t, \nabla f_i(\mathbf{z}_i^t), \mathbf{B}_i^t\}_{i=1}^N$. Then, we elaborate on the scheme for updating the information of the functions.

Note that the second-order approximation of each individual function $f_i(\mathbf{w})$ centered around its current iterate $\mathbf{z}_i^t$

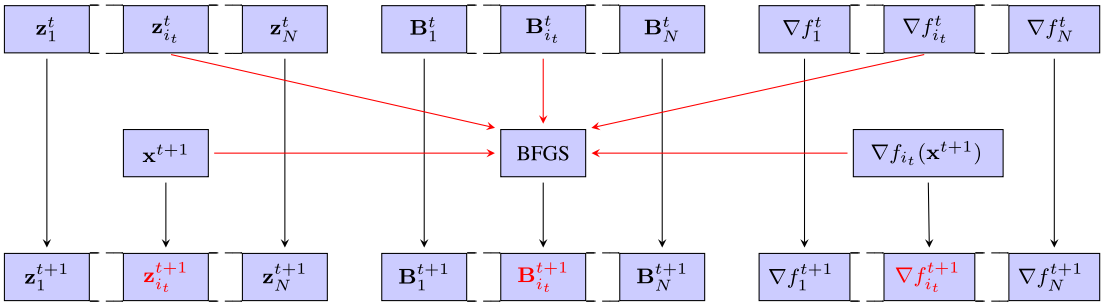



**Fig. 2.** *Updating scheme for variables, gradients, and Hessian approximation matrices of function $f_{i_t}$ at step t. The red arrows indicate the terms used in the update of $\mathbf{B}_{i_t}^{t+1}$ using the BFGS update in (40). The black arrows show the updates of all variables and gradients. The terms $\mathbf{z}_{i_t}^{t+1}$ and $\nabla f_{i_t}^{t+1}$ are updated as $\mathbf{w}^{t+1}$ and $\nabla f_{i_t}(\mathbf{w}^{t+1})$, respectively. All others $\mathbf{z}_j^{t+1}$ and $\nabla f_j^{t+1}$ are set as $\mathbf{z}_j^t$ and $\nabla f_j^t$, respectively.*

is

$$f_i(\mathbf{w}) \approx f_i(\mathbf{z}_i^t) + \nabla f_i(\mathbf{z}_i^t)^T(\mathbf{w}-\mathbf{z}_i^t) + \frac{1}{2}(\mathbf{w}-\mathbf{z}_i^t)^T\nabla^2 f_i(\mathbf{z}_i^t)(\mathbf{w}-\mathbf{z}_i^t). \quad (34)$$

If, for each function $f_i$, we replace the Hessian $\nabla^2 f_i(\mathbf{z}_i^t)$ in (34) by its Hessian approximation $\mathbf{B}_i^t$, then the aggregate function $f(\mathbf{w})$ can be approximated with

$$f(\mathbf{w}) \approx \frac{1}{N}\sum_{i=1}^N \left[f_i(\mathbf{z}_i^t) + \nabla f_i(\mathbf{z}_i^t)^T(\mathbf{w}-\mathbf{z}_i^t) + \frac{1}{2}(\mathbf{w}-\mathbf{z}_i^t)^T\mathbf{B}_i^t(\mathbf{w}-\mathbf{z}_i^t)\right]. \quad (35)$$

Indeed, the right-hand side of (35) is a valid approximation of the aggregate loss $f$, and since it is a quadratic function with respect to $\mathbf{w}$, the right-hand side can be easily optimized. In IQN, the updated variable $\mathbf{w}^{t+1}$ is the solution of the quadratic program in (35), which is

$$\mathbf{w}^{t+1} = \left(\frac{1}{N}\sum_{i=1}^N \mathbf{B}_i^t\right)^{-1}\left[\frac{1}{N}\sum_{i=1}^N \mathbf{B}_i^t\mathbf{z}_i^t - \frac{1}{N}\sum_{i=1}^N \nabla f_i(\mathbf{z}_i^t)\right]. \quad (36)$$

First, note that the update in (36) shows that the updated variable $\mathbf{w}^{t+1}$ is a function of the stored information of all functions $f_1,\ldots,f_n$. Furthermore, we use the aggregated information of variables, gradients, and the quasi-Newton Hessian approximations to evaluate the updated variable. This is done to vanish the noise in approximating both gradients and Hessians as the sequence approaches the optimal argument.

Next, we discuss how the individual gradients and Hessian approximations are updated in IQN once the new variable $\mathbf{w}^{t+1}$ is computed. In each iteration of the IQN method, we update the local information of only a single function, chosen in a cyclic manner. Let $i_t$ be the index of the function selected at time $t$. The variables $\mathbf{z}_i$ are updated according to

$$\mathbf{z}_{i_t}^{t+1} = \mathbf{w}^{t+1}, \quad \mathbf{z}_i^{t+1} = \mathbf{z}_i^t \quad \text{for all } i \neq i_t. \quad (37)$$

The variable corresponding to the updated function $f_{i_t}$ is replaced by the new iterate $\mathbf{w}^{t+1}$, while the remaining variables are simply kept as their previous value. Likewise, we update the table of gradients accordingly with the gradient of $f_{i_t}$ evaluated at the new variable $\mathbf{w}^{t+1}$. The rest of gradients stored in the memory will stay unchanged, that is,

$$\nabla f_{i_t}(\mathbf{z}_i^{t+1}) = \nabla f_{i_t}(\mathbf{w}^{t+1}),\ \nabla f_i(\mathbf{z}_i^{t+1}) = \nabla f_i(\mathbf{z}_i^t), \quad \text{for } i \neq i_t. \quad (38)$$

To the update curvature approximation matrix $\mathbf{B}_{i_t}^t$ associated with the function $f_{i_t}$, we use the steps of BFGS. To do so, we define variable and gradient variations associated with each function $f_i$ as

$$\mathbf{s}_i^t := \mathbf{z}_i^{t+1} - \mathbf{z}_i^t, \qquad \mathbf{y}_i^t := \nabla f_i(\mathbf{z}_i^{t+1}) - \nabla f_i(\mathbf{z}_i^t) \quad (39)$$

respectively. The Hessian approximation $\mathbf{B}_{i_t}^t$ corresponding to $f_{i_t}$ can be computed using the update of BFGS as

$$\mathbf{B}_i^{t+1} = \mathbf{B}_i^t + \frac{\mathbf{y}_i^t\mathbf{y}_i^{tT}}{\mathbf{y}_i^{tT}\mathbf{s}_i^t} - \frac{\mathbf{B}_i^t\mathbf{s}_i^t\mathbf{s}_i^{tT}\mathbf{B}_i^t}{\mathbf{s}_i^{tT}\mathbf{B}_i^t\mathbf{s}_i^t}, \quad \text{for } i = i_t. \quad (40)$$

The Hessian approximation matrices for all other functions remain unchanged, that is, $\mathbf{B}_i^{t+1} = \mathbf{B}_i^t$ for $i \neq i_t$. The system of updates in (37)–(40) explains the mechanism of updating the information of the function $f_{i_t}$ at step $t$. Notice that, to update the Hessian approximation matrix for the $i_t$th function, there is no need to store the variations in (39) since the old variable $\mathbf{z}_i^t$ and gradient $\nabla f_i(\mathbf{z}_i^t)$ are available in memory, and the updated versions $\mathbf{z}_i^{t+1} = \mathbf{w}^{t+1}$ and $\nabla f_i(\mathbf{z}_i^{t+1}) = \nabla f_i(\mathbf{w}^{t+1})$ are evaluated at step $t$ (see Fig. 2).

It is worth noting that, due to the cyclic update rule, the set of iterates $\{\mathbf{z}_1^t, \mathbf{z}_2^t, \ldots, \mathbf{z}_n^t\}$ is equal to the set $\{\mathbf{w}^t, \mathbf{w}^{t-1}, \ldots, \mathbf{w}^{t-n+1}\}$, and hence, the set of variables used in the update of IQN is the set of the last $N$ iterates. This observation shows that the update of IQN in (36) incorporates the information of all the functions $f_1, \ldots, f_n$ to compute the updated variable $\mathbf{w}^{t+1}$; however, it uses delayed variables, gradients, and Hessian approximations rather than the updated variable $\mathbf{w}^{t+1}$ for all functions as in classic quasi-Newton methods. The use of delay allows IQN to update the information of a single function at each iteration, which reduces the computational cost relative to standard quasi-Newton methods.

The update in (36) explains the main logic behind the update of IQN, but it cannot be implemented at a low computational cost, as it requires computation of the sums $\sum_{i=1}^N \mathbf{B}_i^t$, $\sum_{i=1}^N \mathbf{B}_i^t \mathbf{z}_i^t$, and $\sum_{i=1}^N \nabla f_i(\mathbf{z}_i^t)$, as well as the inversion $(\sum_{i=1}^N \mathbf{B}_i^t)^{-1}$. Next, we discuss an efficient implementation of IQN that has a cost of $\mathcal{O}(p^2)$.

### B. Efficient Implementation of IQN

In this section, we show that (36) can be implemented by computing a single gradient and Hessian approximation per iteration. Begin by defining the aggregate Hessian approximation $\tilde{\mathbf{B}}^t := \sum_{i=1}^N \mathbf{B}_i^t$, the aggregate Hessian-variable product $\mathbf{u}^t := \sum_{i=1}^N \mathbf{B}_i^t \mathbf{z}_i^t$, and the aggregate gradient $\mathbf{g}^t := \sum_{i=1}^N \nabla f_i(\mathbf{z}_i^t)$. With these definitions, (36) can be written as

$$\mathbf{w}^{t+1} = (\tilde{\mathbf{B}}^t)^{-1}(\mathbf{u}^t - \mathbf{g}^t). \tag{41}$$

Furthermore, it can be easily verified that the updates for these vectors and matrices can be written as

$$\tilde{\mathbf{B}}^{t+1} = \tilde{\mathbf{B}}^t + \left(\mathbf{B}_{i_t}^{t+1} - \mathbf{B}_{i_t}^t\right) \tag{42}$$

$$\mathbf{u}^{t+1} = \mathbf{u}^t + \left(\mathbf{B}_{i_t}^{t+1}\mathbf{z}_{i_t}^{t+1} - \mathbf{B}_{i_t}^t \mathbf{z}_{i_t}^t\right) \tag{43}$$

$$\mathbf{g}^{t+1} = \mathbf{g}^t + \left(\nabla f_{i_t}\left(\mathbf{z}_{i_t}^{t+1}\right) - \nabla f_{i_t}(\mathbf{z}_{i_t}^t)\right). \tag{44}$$

Given these recursions, it follows that we can keep track of the quantities that are needed in (41) by evaluating only $\mathbf{B}_{i_t}^{t+1}$ and $\nabla f_{i_t}(\mathbf{z}_{i_t}^{t+1})$ at time step $t$. We can, furthermore, avoid the cost of computing $(\tilde{\mathbf{B}}^t)^{-1}$ required in (41) by simplifying the update in (42) to

$$\tilde{\mathbf{B}}^{t+1} = \tilde{\mathbf{B}}^t + \frac{\mathbf{y}_{i_t}^t \mathbf{y}_{i_t}^{tT}}{\mathbf{y}_i^{tT}\mathbf{s}_{i_t}^t} - \frac{\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t \mathbf{s}_{i_t}^{tT} \mathbf{B}_{i_t}^t}{\mathbf{s}_{i_t}^{tT}\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t}. \tag{45}$$

To derive the expression in (45), we substituted the difference $\mathbf{B}_{i_t}^{t+1} - \mathbf{B}_{i_t}^t$ by its rank 2 expression in (40). Given the matrix $(\tilde{\mathbf{B}}^t)^{-1}$, by applying the Sherman–Morrison formula twice to the update in (45), we can compute $(\tilde{\mathbf{B}}^{t+1})^{-1}$ as

$$\left(\tilde{\mathbf{B}}^{t+1}\right)^{-1} = \mathbf{U}^t + \frac{\mathbf{U}^t\left(\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t\right)\left(\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t\right)^T \mathbf{U}^t}{\mathbf{s}_{i_t}^{t\,T}\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t - \left(\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t\right)^T \mathbf{U}^t \left(\mathbf{B}_{i_t}^t \mathbf{s}_{i_t}^t\right)} \tag{46}$$

where the matrix $\mathbf{U}^t$ is evaluated as

$$\mathbf{U}^t = (\tilde{\mathbf{B}}^t)^{-1} - \frac{(\tilde{\mathbf{B}}^t)^{-1}\mathbf{y}_{i_t}^t \mathbf{y}_{i_t}^{tT}(\tilde{\mathbf{B}}^t)^{-1}}{\mathbf{y}_{i_t}^{tT}\mathbf{s}_{i_t}^t + \mathbf{y}_{i_t}^{tT}(\tilde{\mathbf{B}}^t)^{-1}\mathbf{y}_{i_t}^t}. \tag{47}$$

The computational complexity of the updates in (46) and (47) is of the order $\mathcal{O}(p^2)$ rather than the $\mathcal{O}(p^3)$ cost of computing the inverse directly. Hence, the overall cost of IQN is of $\mathcal{O}(p^2)$.

*Remark 6:* The goal of BFGS is to approximate the descent direction of Newton's method using the first-order information. Likewise, in IQN, the goal is to show that the descent directions for all functions $f_1, \ldots, f_N$ are close to the ones for the incremental Newton method. In fact, it has been shown that, in IQN, for each function $f_i$, the Dennis–Moré condition holds [19, Proposition 4]. By exploiting this result, one can show that the sequence of iterates generated by IQN converges to the optimal solution at a superlinear rate [19, Th. 7].

## VII. NUMERICAL EXPERIMENTS

In this section, we numerically compare the convergence properties of RES, oLBFGS, and IQN with the first-order methods and illustrate their advantages in ill-conditioned problems. For problems that the number of samples $N$ is not massive and we can store all gradients and Hessian approximations, we use IQN as it has the best theoretical guarantees among these three methods. For problems that $N$ is large but the number of parameters $p$ is moderate, we use RES as it performs better than oLBFGS that uses a limited memory version of RES. Finally, for problems that both $N$ and $p$ are very large, we use oLBFGS as its memory requirement, and the computational cost per iteration is less than the ones for IQN and RES.

### A. IQN Versus Variance-Reduced First-Order Methods

In this section, we compare the superlinearly convergent IQN method with linearly convergent first-order methods as, similar to IQN, they also use memory to reduce the noise of gradient estimation. In particular, we compare IQN with the SAG algorithm proposed in [10] and its unbiased variant SAGA proposed in [11]. To do so, we focus on quadratic programming. Note that, often, the benefits of second-order methods and quasi-Newton methods are more significant when the objective function behaves similarly to a quadratic function. This is, indeed, not surprising as the main idea of the second-order methods is to approximate the objective function with its quadratic approximation. Consider the following quadratic objective function:

$$\mathbf{w}^* := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \frac{1}{N}\sum_{i=1}^N \left(\frac{1}{2}\mathbf{w}^T\mathbf{A}_i\mathbf{w} + \mathbf{b}_i^T\mathbf{w}\right). \tag{48}$$

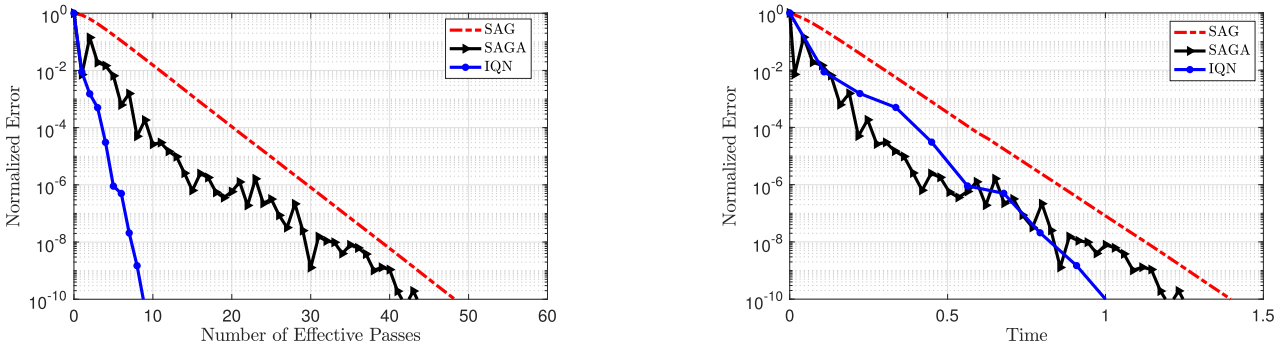



**Fig. 3.** *Comparison of IQN, SAG, and SAGA for a quadratic problem with a small condition number.*

The random matrices $\mathbf{A}_i \in \mathbb{R}^{p\times p}$ are generated such that all of them are PD. Moreover, the vectors $\mathbf{b}_i \in \mathbb{R}^p$ are random vectors for all $i = 1, \ldots, N$. Indeed, when $\{\mathbf{A}_i\}_{i=1}^N$ are all PD, the objective function in (48) is strongly convex. To control the condition number of the problem, we focus on the case that the matrices $\{\mathbf{A}_i\}_{i=1}^N$ are diagonal. To be more precise, consider the case that $\mathbf{A}_i := \text{diag}\{\mathbf{a}_i\}$, where $\mathbf{a}_i$ are random vectors such that the first $p/2$ elements are chosen from $[1, 10^\xi]$ and the remaining $p/2$ elements are selected from $[10^{-\xi}, 1]$. By choosing different values for $\xi$, we can change the condition number of the problem. For instance, if we set $\xi = 1$, then the condition number $\kappa$ of the problem is relatively small, that is, $\kappa = 10^2$, and if we set $\xi = 2$, then the condition number becomes larger; in this case, $\kappa = 10^4$. To ensure that each individual function has a different optimal solution, we choose different values for vectors $\mathbf{b}_i$. These vectors are randomly selected from the box $[0, 10^3]^p$. We set $p = 10$ and $N = 1000$. When we solve a quadratic problem, for any initial point, we are always in the local superlinear convergence neighborhood of IQN, and therefore, its stepsize can be as large as $\eta = 1$.

We first focus on the problem with a small condition number of $\kappa = 10^2$ with $\xi = 1$. In this case, the best choices of stepsize for SAG and SAGA are $\eta = 5 \times 10^{-5}$ and $\eta = 10^{-4}$, respectively, These stepsizes are hand-tuned to obtain the best performance of these first-order variance-reduced methods. The left plot in Fig. 3 showcases the convergence paths of these algorithms' normalized optimal distance error $\|\mathbf{w}^t - \mathbf{w}^*\|/\|\mathbf{w}^0 - \mathbf{w}^*\|$ in terms of number of effective passes over data. As we observe, all the first-order methods converge linearly but at a slower rate compared with IQN. Since the computational complexity of IQN per iteration is higher than these methods, we also compare these algorithms in terms of their runtime (second). This comparison is illustrated in the right plot of Fig. 3. As we observe, the gain of IQN in terms of the runtime is not significant when the problem condition number is small.

Next, we study a problem with a larger condition number. We set $\xi = 2$ that leads to the condition number $\kappa = 10^4$. In this case, the best performance for SAG and SAGA is achieved when the stepsize is $\eta = 2 \times 10^{-4}$ and $\eta = 10^{-4}$, respectively. For this case, we also include the linearly convergent SLBFGS method proposed in [18] and discussed in Section V-C. The results are presented in Fig. 4. As we observe in the left plot of Fig. 4, the gap between IQN and the considered variance-reduced first-order methods, that is, SAG and SAGA, in terms of the number of passes over the data set is even more significant. It is also worth noting that the performance of IQN does not change with the change of condition number. Hence, at least

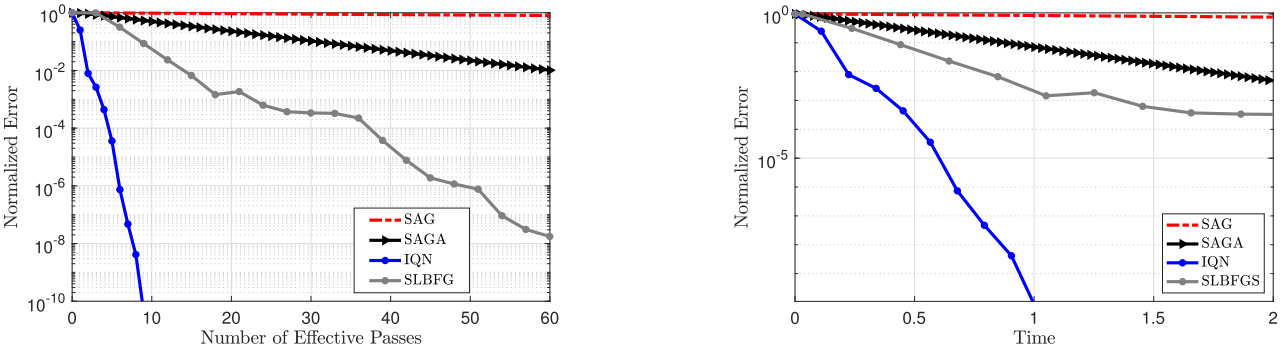



**Fig. 4.** *Comparison of IQN, SLBFGS, SAG, and SAGA for a quadratic problem with a large condition number.*

for the quadratic problem, the IQN method performs independent of the condition number. Indeed, as we expected, the performance of SAG and SAGA substantially degrades compared with the case that the problem has a smaller condition number. The right plot in Fig. 4 also showcases the convergence paths of these methods in terms of runtime. As we observe, in ill-conditioned problems, even in terms of runtime, IQN is significantly faster than the variance-reduced first-order methods. Moreover, note that, both in terms of number passes over data and runtime, IQN outperforms the linearly convergent SLBFGS quasi-Newton method.

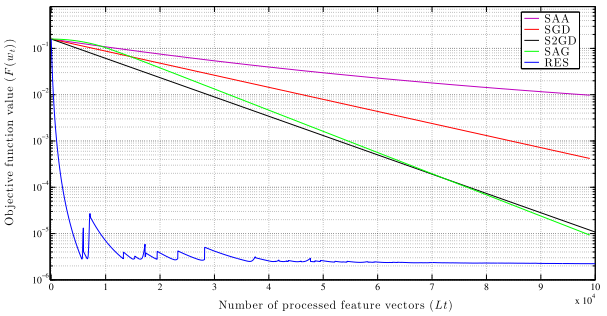



**Fig. 5.** *Comparison of RES, SGD, the SGD accelerations SAA, SAG, and S2GD for an SVM problem.*

## B. RES Versus Stochastic Gradient-Type Methods

In Section VII-A, we focused on the problems that have a reasonable number of samples $N$ so that the cost of storing $N$ matrices for IQN is manageable. However, since the memory required for IQN is $\mathcal{O}(Np^2)$, when the number of samples $N$ and features $p$ are large, IQN requires large memory. In this section, we focus on problems where the number of samples $N$ is large, but the problem dimension $p$ is relatively moderate. In this case, RES is the method of choice as its complexity does not scale with $N$ and its Hessian approximation is more accurate than the one for oLBFGS.

In particular, we study a support vector machine (SVM) problem. In SVM, we aim to find the best hyperplane that separates data points with different labels while achieving the maximum margins. Suppose that we have access to $N$ data points $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^p$ is the $i$th feature vector and $y_i \in \{-1, 1\}$ is its corresponding label. The SVM problem can be written as

$$\mathbf{w}^* := \operatorname*{argmin}_{\mathbf{w}} \ \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \ell((\mathbf{x}_i, y_i); \mathbf{w}) \tag{49}$$

where $\lambda > 0$ is the regularization parameter, and the loss functions $\ell$ measures how well the classifier $\mathbf{w}$ classifies sample $(\mathbf{x}_i, y_i)$. In this experiment, we use the squared hinge loss $\ell((\mathbf{x}, y); \mathbf{w}) = \max(0, 1-y(\mathbf{x}^T\mathbf{w}))^2$. We use a synthetic data set for our experiment. We generate $N = 10^4$ samples of size $p = 400$, where half of them have label 1 and the other half have label $-1$. The components of the samples with label $-1$ are chosen uniformly at random from the interval $[-0.8, 0.2]$, and the elements of those with label 1 are chosen uniformly at random from the interval $[-0.2, 0.8]$. We intentionally enforce an overlap between the two classes so that the data set is not linearly separable. In this experiment, we set the regularization parameter as $\lambda = 10^{-4}$.

We compare RES with the standard SGD algorithm and three other variants of SGD: 1) the SAG algorithm [10] that uses a memory of size $N$ to reduce the noise of gradient estimation; 2) the semi-SGD (S2GD) [14] method that is a hybrid method, meaning that it computes the full gradient once in a while; and 3) the stochastic approximation by averaging (SAA) algorithm [55] that uses the average of all the observed iterates over time.

In this experiment, we use different batch sizes for different algorithms. Hence, to have a fair comparison, we compare the algorithms in terms of the number of processed feature vectors $bt$, which is equal to the product of batch size and the number of iterations. For RES, we set the batch size as $b = 20$, and the stepsize is selected as $\eta_t = \eta_0 T_0/(T_0 + t)$, where $\eta_0 = 10^{-1}$ and $T_0 = 10$. For SGD, S2GD, and SAA, we select the parameters that achieve optimal performance after processing $10^4$ feature vectors. Fig. 5 demonstrates the convergence paths of these algorithms in terms of the number of processed feature vectors. As we observe, RES converges significantly faster than the other studied methods as the problem is ill-conditioned. Note that since the computation cost per iteration of RES is higher than the considered first-order methods, we also compare their CPU runtime for achieving a specific accuracy. In Table 1, we report the CPU runtime of these algorithms to achieve the objective function value of $10^{-4}$. Indeed, the advantage of RES with respect to SGD, S2GD, and SAA is more significant when we compare them in terms of the number of processed samples and it downgrades when we compare their CPU runtimes.

## C. oLBFGS Versus SGD

Next, we focus on a large-scale learning problem where both the number of samples $N$ and problem dimension $p$ are extremely large. In this setting, IQN and RES become impractical since their computational cost per iteration is not linear in $p$. Also, methods such as SAG, SAGA, and S2GD are not practical as they either require memory of size $Np$ or computation of the full gradient once in a while. Hence, the only affordable choices for this setting

**Table 1** CPU Runtimes of RES, SGD, the SGD Accelerations SAA, SAG, and S2GD to Achieve the Cost $F(\mathbf{w}_t) = 10^{-4}$

| | RES | SGD | SAG | S2GD | SAA |
|---|---|---|---|---|---|
| CPU runtime | 0.8 s | $>$ 1.5 s | 1.3 s | 1.2 s | $>$ 1.7 s |

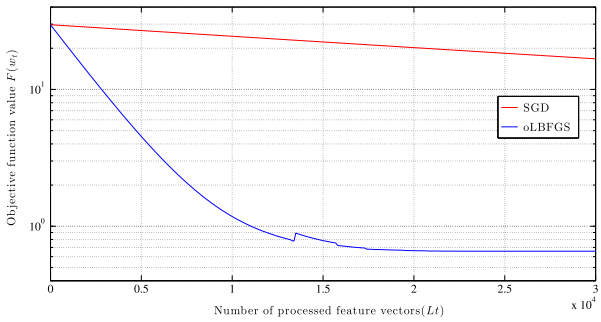



**Fig. 6.** ***Comparison of oLBFGS and SGD for a large-scale CTR prediction problem.***

are the standard SGD method and the oLBFGS algorithm that requires a memory of $\mathcal{O}(p)$ (independent of $N$), and its computation cost per iteration is also $\mathcal{O}(p)$ as in SGD.

We focus on a click-through rate (CTR) prediction problem that appears in online advertising. In this case, the feature vectors are created based on the information of users, the query that they have searched, and the advertisements that have been shown to the users. The goal is to use this information to predict the advertisement that has the highest probability to be clicked for a given user and a specific query. More details on how the feature vectors are generated can be found in [8]. The data set that we use is the Tencent search engine data set [56] that contains the outcomes of 236 million searches. Each feature vector has 174 026 elements, and the label of each feature vector is 1 if the search instance corresponds to a case that the user has clicked on the ad and it is $-1$ otherwise. Note that we expect this problem to be highly ill-conditioned as the feature vector components are significantly different from each other in terms of value and range. Hence, the gain by using stochastic quasi-Newton methods should be substantial. We solve this problem using a logistic regression model. To do so, we assume that the CTR that is the probability of observing $y = 1$ is given by

$$\text{CTR}(\mathbf{x};\mathbf{w}) := \text{P}\left[y = 1 \,\middle|\, \mathbf{x};\mathbf{w}\right] = \frac{1}{1+\exp\left(-\mathbf{x}^T\mathbf{w}\right)} \tag{50}$$

and we have $\text{P}\left[y = -1 \,\middle|\, \mathbf{x};\mathbf{w}\right] = 1 - \text{P}\left[y = 1 \,\middle|\, \mathbf{x};\mathbf{w}\right]$. The problem of finding the best predictor $\mathbf{w}^*$ can be written as

$$\mathbf{w}^* := \operatorname{argmin} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \log\left(1+\exp\left(-y_i\mathbf{x}_i^T\mathbf{w}\right)\right).$$

Out of the $236 \times 10^6$ samples in the data set, we randomly choose $10^6$ samples to use as the training set. We also set the regualrization parameter $\lambda$ as $\lambda = 10^{-6}$. The stepsizes for both algorithms are of the form $\eta_t = \eta_0 T_0/(T_0 + t)$. We select the stepsize parameters for oLBFGS as $\eta_0 = 10^{-2}$ and $T_0 = 10^4$ and SGD as $\eta_0 = 10^{-1}$ and $T_0 = 10^6$. Moreover, the batch size for SGD is $b = 20$, while the batch size for oLBFGS is $b = 100$. These parameters are handtuned to observe the best performance. We further set the memory size for the oLBFGS method as $\tau = 10$.

The convergence paths of these methods in terms of objective function value versus the number of processed features are illustrated in Fig. 6. As we observe, in this case, oLBFGS converges significantly faster than SGD. This example again showcases the advantage of stochastic quasi-Newton methods against stochastic first-order methods when we deal with large-scale problems that often have a large condition number.

## VIII. CONCLUSION

In this article, we reviewed four quasi-Newton methods for solving stochastic optimization problems. We started by reviewing RES that uses stochastic gradients instead of gradients and modifies the update of BFGS to ensure that the eigenvalues of the Hessian approximation matrices stay bounded. Then, we discussed oLBFGS that aims at reducing the computational cost of RES by using a limited memory scheme that only uses the curvature information of a small number of recent iterates. We further mentioned that both RES and oLBFGS obtain a sublinear rate in stochastic settings as they need to use diminishing stepsizes to control the noise of gradient estimation. We then discussed SLBFGS that achieves a linear convergence rate by reducing the noise of gradient estimation. Finally, we studied IQN that in finite-sum settings converges superlinearly by using variance reduction techniques and a proper quadratic approximation of individual functions. Next, we briefly mention a few future research directions on stochastic quasi-Newton methods.

### A. Memory Efficiency

As shown in Fig. 2, IQN requires access to the most recent version of variables, gradients, and the Hessian approximations for all $N$ individual functions to reduce the noise of gradient and the Hessian approximations. As a result, the memory required for running IQN is $\mathcal{O}(Np^2)$ as we need to store $N$ matrices of size $p^2$. Indeed, this could be a crucial issue when we plan to use IQN for solving problems with many parameters (large $p$) or many samples (large $N$). Therefore, a memory-efficient implementation of the IQN method could be an interesting research problem to explore.

### B. Nonconvex Setting

A vast majority of theories for quasi-Newton methods and their stochastic (incremental) variants are developed for convex functions. A fundamental challenge in designing quasi-Newton methods for nonconvex settings is that the objective function Hessian is not PD in nonconvex problems, and as a result, the Hessian approximations of quasi-Newton methods may have nonnegative eigenvalues and, in some cases, may not be invertible. To resolve this issue state-of-the-art methods, ensure that the direction

$\mathbf{B}_t^{-1}\nabla F(\mathbf{w}_t)$ is a valid descent direction by preserving the positive definiteness of the Hessian approximations [52], [57]. However, a proper approximation of curvature while enforcing PD-ness of Hessian approximations is not possible as, in nonconvex settings, the Hessian may not be PD. Hence, developing a convergent quasi-Newton method for nonconvex settings that properly approximate the curvature is of interest.

### C. Adaptive Sample Size Learning

Stochastic methods are suitable for the setting in which storing data is infeasible due to memory limitations. Advances in cloud computing, however, allow for storing and fetching large data sets for offline problems. Hence, an alternative approach for solving ERM problems is the adaptive sample size strategy in which we do not use a partition of the training set as in the case of stochastic methods but a nested collection of subsets that we grow geometrically [58]–[60]. The main idea here is to increase the size of the training set such that the approximate solution for the current ERM problem stays close to the solution of the next problem with more samples, and hence, the new problem can be solved quickly. Studying the case that we use quasi-Newton methods for solving subproblems is interesting, as they converge superlinearly and can solve the subproblems quickly when the initial iterate is close to the solution.

### D. Nonasymptotic Superlinear Convergence Rate

Most classic superlinear convergence results for deterministic quasi-Newton methods are asymptotic. However, in a recent line of work, nonasymptotic superlinear convergence of DFP and BFGS has been established [61], [62]. In particular, it has been shown that both these algorithms converge to the optimal solution at a superlinear rate of $\mathcal{O}((1/t)^{t/2})$, where $t$ is the number of iterations. This result shows that, at least locally, the quasi-Newton methods are provably faster than first-order methods in deterministic settings. Establishing a similar result for the stochastic setting would be an interesting research direction. ■

## REFERENCES


[1] C. G. Broyden, "The convergence of single-rank quasi-Newton methods," *Math. Comput.*, vol. 24, no. 110, pp. 365–382, 1970.

[2] R. Fletcher, "A new approach to variable metric algorithms," *Comput. J.*, vol. 13, no. 3, pp. 317–322, Mar. 1970.

[3] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Math. Comput.*, vol. 24, no. 109, pp. 23–26, 1970.

[4] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1970.

[5] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, Sep. 2009.

[6] A. Mokhtari and A. Ribeiro, "RES: Regularized stochastic BFGS algorithm," *IEEE Trans. Signal Process.*, vol. 62, no. 23, pp. 6089–6104, Dec. 2014.

[7] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2007, pp. 436–443.

[8] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory BFGS," *J. Mach. Learn. Res.*, vol. 16, pp. 3151–3181, 2015.

[9] N. Le Roux, M. Schmidt, and F. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2663–2671.

[10] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Math. Program.*, vol. 162, nos. 1–2, pp. 83–112, Mar. 2017.

[11] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1646–1654.

[12] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 315–323.

[13] D. Blatt, A. O. Hero, and H. Gauchman, "A convergent incremental gradient method with a constant step size," *SIAM J. Optim.*, vol. 18, no. 1, pp. 29–51, Jan. 2007.

[14] J. Konecnỳ and P. Richtárik, "Semi-stochastic gradient descent methods," vol. 2, nos. 2–1, p. 3, 2013, *arXiv:1312.1666*. [Online]. Available: https://arxiv.org/abs/1312.1666

[15] L. Zhang, M. Mahdavi, and R. Jin, "Linear convergence with condition number independent access of full gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 980–988.

[16] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo, "On the convergence rate of incremental aggregated gradient algorithms," *SIAM J. Optim.*, vol. 27, no. 2, pp. 1035–1048, 2017.

[17] A. Mokhtari, M. Gürbüzbalaban, and A. Ribeiro, "Surpassing gradient descent provably: A cyclic incremental method with linear convergence rate," *SIAM J. Optim.*, vol. 28, no. 2, pp. 1420–1447, Jan. 2018.

[18] P. Moritz, R. Nishihara, and M. I. Jordan, "A linearly-convergent stochastic L-BFGS algorithm," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2016, pp. 249–258.

[19] A. Mokhtari, M. Eisen, and A. Ribeiro, "IQN: An incremental quasi-Newton method with local superlinear convergence rate," *SIAM J. Optim.*, vol. 28, no. 2, pp. 1670–1698, Jan. 2018.

[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Int. Conf. Comput. Statist. (COMPSTAT)*. Paris, France: Physica-Verlag, Aug. 2010, pp. 177–186, doi: 10.1007/978-3-7908-2604-3_16.

[21] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for SVM," *Math. Program.*, vol. 127, no. 1, pp. 3–30, Mar. 2011.

[22] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2004, p. 116.

[23] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM J. Optim.*, vol. 19, no. 4, pp. 1574–1609, Jan. 2009.

[24] A. R. Conn, N. I. M. Gould, and P. L. Toint, "Convergence of quasi-Newton matrices generated by the symmetric rank one update," *Math. Program.*, vol. 50, nos. 1–3, pp. 177–195, Mar. 1991.

[25] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Math. Comput.*, vol. 19, no. 92, pp. 577–593, 1965.

[26] C. G. Broyden, J. E. Dennis, and J. J. Moré, "On the local and superlinear convergence of quasi-Newton methods," *IMA J. Appl. Math.*, vol. 12, no. 3, pp. 223–245, 1973.

[27] D. M. Gay, "Some convergence properties of Broyden's method," *SIAM J. Numer. Anal.*, vol. 16, no. 4, pp. 623–630, 1979.

[28] W. C. Davidon, "Variable metric method for minimization," *SIAM J. Optim.*, vol. 1, no. 1, pp. 1–17, 1991.

[29] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Comput. J.*, vol. 6, no. 2, pp. 163–168, Aug. 1963.

[30] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Math. Comput.*, vol. 35, no. 151, pp. 773–782, 1980.

[31] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, nos. 1–3, pp. 503–528, Aug. 1989.

[32] J. J. Moré and J. A. Trangenstein, "On the global convergence of Broyde's method," *Math. Comput.*, vol. 30, no. 135, pp. 523–540, 1976.

[33] M. J. D. Powell, "On the convergence of the variable metric algorithm," *IMA J. Appl. Math.*, vol. 7, no. 1, pp. 21–36, 1971.

[34] J. E. Dennis and J. J. Moré, "A characterization of superlinear convergence and its application to quasi-Newton methods," *Math. Comput.*, vol. 28, no. 126, pp. 549–560, 1974.

[35] R. H. Byrd, J. Nocedal, and Y.-X. Yuan, "Global convergence of a class of quasi-Newton methods on convex problems," *SIAM J. Numer. Anal.*, vol. 24, no. 5, pp. 1171–1190, 1987.

[36] W. Gao and D. Goldfarb, "Quasi-Newton methods: Superlinear convergence without line searches for self-concordant functions," *Optim. Methods Softw.*, vol. 34, no. 1, pp. 194–217, Jan. 2019.

[37] A. Griewank and P. L. Toint, "Local convergence analysis for partitioned quasi-Newton updates," *Numerische Math.*, vol. 39, no. 3, pp. 429–448, Oct. 1982.

[38] J. E. Dennis, H. J. Martinez, and R. A. Tapia, "Convergence theory for the structured BFGS secant method with an application to nonlinear least squares," *J. Optim. Theory Appl.*, vol. 61, no. 2, pp. 161–178, May 1989.

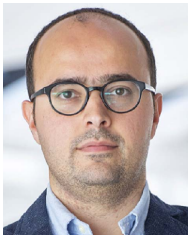
[39] Y.-X. Yuan, "A modified BFGS algorithm for unconstrained optimization," *IMA J. Numer. Anal.*, vol. 11, no. 3, pp. 325–332, 1991.

[40] L. Qi, "On superlinear convergence of quasi-Newton methods for nonsmooth equations," *Oper. Res. Lett.*, vol. 20, no. 5, pp. 223–228, Jun. 1997.

[41] M. Al-Baali, "Global and superlinear convergence of a restricted class of self-scaling methods with inexact line searches, for convex functions,"

*Comput. Optim. Appl.*, vol. 9, no. 2, pp. 191–203, 1998.
[42] D. Li and M. Fukushima, "A globally and superlinearly convergent Gauss-Newton-based BFGS method for symmetric nonlinear equations," *SIAM J. Numer. Anal.*, vol. 37, no. 1, pp. 152–172, Jan. 1999.
[43] H. Yabe, H. Ogasawara, and M. Yoshino, "Local and superlinear convergence of quasi-Newton methods based on modified secant conditions," *J. Comput. Appl. Math.*, vol. 205, no. 1, pp. 617–632, Aug. 2007.
[44] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2006.
[45] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-Newton method for large-scale optimization," *SIAM J. Optim.*, vol. 26, no. 2, pp. 1008–1031, Jan. 2016.
[46] A. Lucchi, B. McWilliams, and T. Hofmann, "A variance reduced stochastic Newton method," 2015, *arXiv:1503.08316*. [Online]. Available: http://arxiv.org/abs/1503.08316
[47] R. Gower, D. Goldfarb, and P. Richtárik, "Stochastic block BFGS: Squeezing more curvature out of data," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1869–1878.
[48] F. Yousefian, A. Nedić, and U. V. Shanbhag, "Stochastic quasi-Newton methods for non-strongly convex problems: Convergence and rate analysis," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, Dec. 2016, pp. 4496–4503.
[49] A. Jalilzadeh, A. Nedić, U. V. Shanbhag, and F. Yousefian, "A variable sample-size stochastic quasi-Newton method for smooth and nonsmooth stochastic convex optimization," in *Proc. IEEE Conf. Decis. Control (CDC)*, Dec. 2018, pp. 4097–4102.
[50] A. S. Berahas, J. Nocedal, and M. Takác, "A multi-batch L-BFGS method for machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1055–1063.
[51] S. Soori, K. Mishchenko, A. Mokhtari, M. M. Dehnavi, and M. Gurbuzbalaban, "DAve-QN: A distributed averaged quasi-Newton method with local superlinear convergence rate," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 1965–1976.
[52] X. Wang, S. Ma, D. Goldfarb, and W. Liu, "Stochastic quasi-Newton methods for nonconvex stochastic optimization," *SIAM J. Optim.*, vol. 27, no. 2, pp. 927–956, Jan. 2017.
[53] A. S. Berahas, M. Jahani, and M. Takáč, "Quasi-Newton methods for deep learning: Forget the past, just sample," 2019, *arXiv:1901.09997*. [Online]. Available: http://arxiv.org/abs/1901.09997
[54] X. Wang, X. Wang, and Y.-X. Yuan, "Stochastic proximal quasi-Newton methods for non-convex composite optimization," *Optim. Methods Softw.*, vol. 34, no. 5, pp. 922–948, Sep. 2019.
[55] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, Jul. 1992.
[56] G. Sun. (2012). *KDD Cup Track 2 soso.com ADS Prediction Challenge*. Accessed: Aug. 1, 2012. [Online]. Available: https://www.kdd.org/kdd-cup/view/kdd-cup-2012-track-2
[57] D.-H. Li and M. Fukushima, "On the global convergence of the BFGS method for nonconvex unconstrained optimization problems," *SIAM J. Optim.*, vol. 11, no. 4, pp. 1054–1064, Jan. 2001.
[58] A. Mokhtari, H. Daneshmand, A. Lucchi, T. Hofmann, and A. Ribeiro, "Adaptive Newton method for empirical risk minimization to statistical accuracy," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4062–4070.
[59] A. Mokhtari and A. Ribeiro, "First-order adaptive sample size methods to reduce complexity of empirical risk minimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2060–2068.
[60] M. Eisen, A. Mokhtari, and A. Ribeiro, "Large scale empirical risk minimization via truncated adaptive Newton method," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 1447–1455.
[61] A. Rodomanov and Y. Nesterov, "Rates of superlinear convergence for classical quasi-Newton methods," 2020, *arXiv:2003.09174*. [Online]. Available: http://arxiv.org/abs/2003.09174
[62] Q. Jin and A. Mokhtari, "Non-asymptotic superlinear convergence of standard quasi-Newton methods," 2020, *arXiv:2003.13607*. [Online]. Available: http://arxiv.org/abs/2003.13607

## ABOUT THE AUTHORS



**Aryan Mokhtari** (Member, IEEE) received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2011, the M.Sc. and Ph.D. degrees in electrical and systems engineering from the University of Pennsylvania (Penn), Philadelphia, PA, USA, in 2014 and 2017, respectively, and the A.M. degree in statistics from the Wharton School, Penn, in 2017.

He was a Research Fellow with the Simons Institute for the Theory of Computing, Berkeley, CA, USA. He was a Postdoctoral Associate with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA. His research interests include the areas of optimization, machine learning, and signal processing. His current research focuses on the theory and applications of convex and nonconvex optimization in large-scale machine learning problems.

Dr. Mokhtari was a recipient of the Penn's Joseph and Rosaline Wolf Award for the Best Doctoral Dissertation in electrical and systems engineering and the Simons-Berkeley Fellowship.

**Alejandro Ribeiro** (Member, IEEE) received the B.Sc. degree in electrical engineering from the Universidad de la Republica Oriental del Uruguay, Montevideo, Uruguay, in 1998, and the M.Sc. and Ph.D. degrees in electrical engineering from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, in 2005 and 2007, respectively.

From 1998 to 2003, he was a member of the Technical Staff at Bellsouth Montevideo, Montevideo, Uruguay. After his M.Sc. and Ph.D. studies, in 2008, he joined the University of Pennsylvania (Penn), Philadelphia, PA, USA, where he is currently the Rosenbluth Associate Professor with the Department of Electrical and Systems Engineering. His research interests include the applications of statistical signal processing to the study of networks and networked phenomena. His focus is on structured representations of networked data structures, graph signal processing, network optimization, robot teams, and networked control.

Dr. Ribeiro is a Fulbright Scholar and a Penn Fellow. He received the 2014 O. Hugo Schuck Best Paper Award, the 2012 S. Reid Warren, Jr., Award presented by the Penn's Undergraduate Student Body for outstanding teaching, the NSF CAREER Award in 2010, and the best paper awards at the 2016 SSP Workshop, the 2016 SAM Workshop, the 2015 Asilomar SSC Conference, The American Control Conference (ACC) 2013, and The International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2005 and 2006.