Faster Stochastic Quasi-Newton Methods

Qingsong Zhang, Feihu Huang, Cheng Deng, Senior Member, IEEE, and Heng Huang

Abstract—Stochastic optimization methods have become a class of popular optimization tools in machine learning. Especially, stochastic gradient descent (SGD) has been widely used for machine learning problems, such as training neural networks, due to low per-iteration computational complexity. In fact, the Newton or quasi-newton (QN) methods leveraging the second-order information are able to achieve a better solution than the first-order methods. Thus, stochastic QN (SQN) methods have been developed to achieve a better solution efficiently than the stochastic first-order methods by utilizing approximate second-order information. However, the existing SQN methods still do not reach the best known stochastic first-order oracle (SFO) complexity. To fill this gap, we propose a novel faster stochastic QN method (SpiderSQN) based on the variance reduced technique of SIPDER. We prove that our SpiderSQN method reaches the best known SFO complexity of $\mathcal{O}(n + n^{1/2} \varepsilon^{-2})$ in the finite-sum setting to obtain an $\varepsilon$-first-order stationary point. To further improve its practical performance, we incorporate SpiderSQN with different momentum schemes. Moreover, the proposed algorithms are generalized to the online setting, and the corresponding SFO complexity of $\mathcal{O}(\varepsilon^{-3})$ is developed, which also matches the existing best result. Extensive experiments on benchmark data sets demonstrate that our new algorithms outperform state-of-the-art approaches for nonconvex optimization.

Index Terms—Momentum acceleration, nonconvex optimization, stochastic quasi-Newton (SQN) method, variance reduction.

I. INTRODUCTION

In this article, we focus on the following unconstrained stochastic nonconvex optimization:

$$\min_{x \in \mathbb{R}^d} f(x) := \begin{cases} \mathbb{E}_{u \sim \mathcal{D}}[f_u(x)], & \text{(online)} \\ \frac{1}{n} \sum_{i=1}^{n} f_i(x), & \text{(finite-sum)} \end{cases} \quad (P)$$

Manuscript received April 25, 2020; revised October 5, 2020; accepted January 15, 2021. The work of Qingsong Zhang and Cheng Deng was supported in part by the National Natural Science Foundation of China under Grant 62073161, in part by the National Key Research and Development Program of China under Grant 2017YFE0104100, and in part by the China Research Project under Grant 6141B07270429 and Grant 61806093. (Corresponding author: Heng Huang.)

Qingsong Zhang is with the School of Electronic Engineering, Xidian University, Xi’an 710071, China, and also with JD Tech, LLC, Beijing 100176, China (e-mail: qszhang1995@gmail.com).

Feihu Huang is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: huangfeihu2018@gmail.com; feh23@pitt.edu).

Cheng Deng is with the School of Electronic Engineering, Xidian University, Xi’an 710071, China (e-mail: chdeng@mail.xidian.edu.cn).

Heng Huang is with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA, and also with JD Finance American Corporation, Mountain View, CA 94043 USA (e-mail: heng.huang@pitt.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3056947.

Digital Object Identifier 10.1109/TNNLS.2021.3056947

where $x \in \mathbb{R}^d$ corresponds to the parameters defining a model, $\mathbb{E}_{u \sim \mathcal{D}}[f_u(x)]$ denotes a population risk over $u \sim \mathcal{D}$, and $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ denotes the loss on the $i$th sample for $\forall i \in 1, \ldots, n$ (or $i \sim \mathcal{D}$). Problem (1) capsules a wide range of machine learning problems, such as truncated square loss [1] for regression and deep neural network [2]. In fact, the stochastic gradient descent (SGD) [3] is a representative method to solve the problem (1) due to its per-iteration computation efficiency. Recently, there have been many works studying SGD and its variance reduction variants, including stochastic variance reduced gradient (SVRG) [4], SAGA [5], stochastically controlled stochastic gradient (SCSG) [6], stochastic recursive gradient algorithm (SARAH) [7], stochastic nested variance-reduced gradient (SNVRG) [8], and SPIIDER [9, 10]. In particular, SPIIDER has been shown in [9] to achieve the stochastic first-order oracle (SFO) complexity lower bound for a certain regime. Such idea has been extended to optimization over mainfolds in [11], zeroth-order optimization in [12] and [13], cubic-regularized method in [14], and alternating direction method of multipliers in [15].

Although SGD is very effective, its performance may be poor owing that it only utilizes first-order information. In contrast, Newton’s method utilizing the Hessian information is more robust and can achieve better accuracy [16, 17], while it is extremely time consuming to compute the Hessian matrix and its inverse. Therefore, many works have been proposed toward designing better SGD methods integrated with approximate Hessian information, i.e., the SQN methods. There have been many works focusing on developing SQN methods, such as SGD with quasi-Newton (SGD-QN) studied in [18] and stochastic approximation-based L-BFGS proposed in [19]. Recently, some SQN methods equipped with the variance reduction technique have been developed to alleviate the effect of variance introduced by stochastic estimator [20–23]. Besides the above-mentioned methods concerning convex or strongly convex problems, progresses have been made toward designing SQN methods for nonconvex cases. Wang et al. [24] analyzed the convergence guarantee of the SGD-QN for nonconvex problems, Wang et al. [25] developed a stochastic proximal QN for nonconvex composite optimization, and Gao et al. [26] proposed the stochastic L-BFGS method for nonconvex sparse learning problems.

Stochastic QN (SQN) methods inherit many appealing advantages from both SGD and QN methods, e.g., efficiency, robustness, and better accuracy. However, existing SQN methods still do not reach the best known SFO complexity, resulting in the limited application to machine learning. It is, thus, of vital importance to improve the SFO complexity of SQN methods for nonconvex optimization. For this reason, we propose a faster SQN method (namely, SpiderSQN) by leveraging the variance reduction technique of SIPDER.
To improve the practical performance of the proposed methods, we apply momentum schemes to them, which are demonstrated to have satisfactory practical effects. Moreover, we prove that our SpiderSQN-M algorithms are extended to the online case with a theoretical guarantee. To give a thorough comparison of our proposed algorithm with existing stochastic first-order algorithms and SQN for nonconvex optimization, we summarize the SFO complexity of the most relevant algorithms to achieve an $\epsilon$-first-order stationary point in Table I. The main contributions of this article are summarized as follows.

1) We propose a novel faster SQN method (SpiderSQN) for nonconvex optimization in the form of finite sum. Moreover, we prove that the SpiderSQN can achieve the best known optimal SFO complexity of $O(n_{\epsilon}^{-2})$ to obtain an $\epsilon$-first-order stationary point.

2) We extend the SpiderSQN to the online setting and propose the faster online SpiderSQN algorithms for nonconvex optimization. Moreover, the online SpiderSQN achieve the best known optimal SFO complexity of $O(\epsilon^{-3})$.

3) To improve the practical performance of the proposed methods, we apply momentum schemes to them, which are demonstrated to have satisfactory practical effects.

4) Moreover, we prove that our SpiderSQN-M algorithms have the lower SFO complexity of $O(n_{\epsilon}^{-2})$, which achieves the optimal SFO complexity of $O(n_{\epsilon}^{-2})$.

## II. Preliminaries

In this section, some preliminaries are presented. Since finding the global minimum of problem (1) is general NP-hard [27], this work instead focuses on finding an $\epsilon$-first-order stationary point and studies the SFO complexity of achieving it. First, we give the necessary definitions and assumptions.

**Definition 1**: An $\epsilon$-first-order stationary point denotes that for $x$ uniformly drawn from $x_1, \ldots, x_K$, where $K$ is the total number of iterations there is $E\|\nabla f(x)\| \leq \epsilon$, where $\epsilon > 0$ is the accuracy parameter.
B. Newton’s Methods for Nonconvex Optimization

Newton’s methods using the Hessian information have a rapid convergence rate (both in theory and practice) [22] and are popular for solving nonconvex problems [31–33]. However, the time consumption of computing the Hessian matrix and its inverse is extremely high. To address this problem, many QN-based methods have been widely studied, such as BFGS, L-BFGS, and the damped L-BFGS [34]. In this article, we adopt the stochastic damped L-BFGS (SdLBFGS) [24] for nonconvex optimization. Let $k$ be current iteration, based on history information, SdLBFGS uses a two-loop recursion to generate a descent direction $d_k = H_k v_k$ without calculating inverse matrix $H_k$ explicitly. Specially, at step 1, vector pair $(s_{k-1}, \hat{y}_{k-1})$ is computed as $s_{k-1} = x_k - x_{k-1}$ and $\hat{y}_{k-1} = v_k - \nu_{k-1}$, and $\gamma_k = \max((\hat{y}_{k-1}^T \hat{y}_{k-1}) / s_{k-1}^T s_{k-1}, 1)$, where $\delta$ is a positive constant. At step 2, SdLBFGS introduces a vector $\hat{y}_{k-1}$

$$\hat{y}_{k-1} = \theta_{k-1} \hat{y}_{k-1} + (1 - \theta_{k-1}) H_{k,0}^{-1} s_{k-1}, k \geq 1$$

(5)

where $H_{k,0} = H_k^{-1} I_{d \times d}$, $k \geq 0$, and $\theta_{k-1}$ is defined as

$$\theta_{k-1} = \begin{cases} 0.75 \sigma_{k-1}, & \text{if } s_{k-1}^T \hat{y}_{k-1} < 0.25 \sigma_{k-1} \\ \sigma_{k-1} - s_{k-1}^T \hat{y}_{k-1}, & \text{otherwise} \end{cases}$$

(6)

where $\sigma_{k-1} = s_{k}^T H_{k,0}^{-1} s_{k-1}$. Based on $(s_{k-1}, \hat{y}_{k-1})$, $H_k v_k$ can be approximated through steps 3–10.

Importantly, SdLBFGS is a computational efficient program because the whole procedure takes only $(6m + 6d)$ multiplications. Especially, the SdLBFGS with variance reduction is proposed [24] by incorporating SdLBFGS into SVRG. However, its best SFO complexity to obtain an $\epsilon$-first-order stationary point is $O(n^{2/3} \epsilon^{-2})$, which is not competitive to state-of-the-art stochastic first-order methods. Therefore, it is desirable to improve the SFO complexity of existing SQN methods.

C. Momentum Acceleration for Nonconvex Optimization

Momentum acceleration scheme is a simple but widely used acceleration technique for an optimization problem. Recently, a variety of accelerated methods have been developed for nonconvex optimization. For example, the stochastic gradient algorithms with momentum scheme are proposed in [28], which have been proved to converge as fast as the gradient descent method for nonconvex problems. Li et al. [35] explored the convergence of the algorithm proposed in [36] under a certain local gradient dominance geometry for nonconvex optimization. Furthermore, Wang et al. [37] studied the convergence to a second-order stationary point under the momentum scheme. However, existing works hardly ever study the acceleration of the SQN method for nonconvex optimization. To this end, this article focuses on accelerating SQN methods with different momentum schemes.

III. FASTER SQN METHODS FOR NONCONVEX OPTIMIZATION

In this section, we propose a novel faster SQN method to solve the nonconvex problem (1) for finite-sum case.

A. Spider Stochastic Quasi-Newton Algorithm

To improve the SFO complexity of the SQN method, a new variance reduction technique SPIDER/SpiderBoost is adopted to control its intrinsic variance. The proposed SpiderSQN with improved SFO complexity is shown in Algorithm 2.

At each iteration, besides evaluating the full gradient every $q$ iterations, the stochastic gradient $v_k$ is updated as

$$v_k = \nabla f_{\zeta_k}(x_k) - \nabla f_{\zeta_k}(x_{k-1}) + v_{k-1}$$

(7)

where $\nabla f_{\zeta_k}(x_k) = (1/|\zeta_k|) \sum_{i \in \zeta_k} \nabla f_i(x_k)$ and $\zeta_k$ is a mini-batch where samples are uniformly sampled with replacement. It is obvious from (7), a more fresh stochastic gradient information $v_{k-1}$ is utilized to update $v_k$, and thus, SpiderSQN has an improved SFO complexity compared with existing SQN methods. At step 8, $x_k$ is updated by the Hessian informative descent direction.

B. Spider Stochastic Quasi-Newton With Momentum Scheme

To improve the practical performance of SpiderSQN, the momentum scheme is adopted for acceleration. The framework of SpiderSQN with momentum scheme (referred to as SpiderSQNM) is shown in Algorithm 3. The momentum
scheme in Algorithm 3 refers to steps 4, 11, and 12, where variables \( x_k \) and \( y_k \) are updated through the \( d_k \), and \( z_k \) is a convex combination of \( x_k \) and \( y_k \) controlled by the momentum coefficient \( \alpha_k \). In this algorithm, an iterationwise diminishing scheme is applied, where the momentum coefficient is set as \( \alpha_k = 2/k + 1 \).

C. Other Momentum Acceleration Strategies

The momentum scheme adopted in Algorithm 3 is a vanilla one whose momentum coefficient \( \alpha_k \) is iterationwise diminishing. When the iteration \( k \) becomes larger, \( \alpha_k \) can be considerably small, leading to a limited acceleration. Thus, other momentum acceleration strategies are explored to alleviate this problem. Following are two powerful momentum schemes, where \( \alpha_k \) can remain relatively large after many epochs. One is the epochwise-restart scheme, whose \( \alpha_k \) is set as

\[
\alpha_k = \frac{2}{\text{mod}(k, q) + 1}, \quad k = 0, \ldots, K - 1.
\]

As the name suggests, \( \alpha_k \) restarts at the beginning of each epoch. Another effective momentum strategy is the epochwise-diminishing scheme with following momentum coefficient

\[
\alpha_k = \frac{2}{k + q}, \quad k = 0, \ldots, K - 1
\]

where \( \lfloor \cdot \rfloor \) denotes the ceiling function. As defined in (9), the momentum coefficient \( \alpha_k \) is a constant during a fixed epoch and will diminish slowly as \( k \) growing sharply. To obtain the variants of SpiderSQN with the above-mentioned two momentum schemes, one just replace the \( \alpha_k \) in Algorithm 3 as defined.

IV. FASTER SQN METHODS FOR ONLINE NONCONVEX OPTIMIZATION

In super large-scale learning, sample size \( n \) can be considerably large or even infinite. It is, thus, desirable to design algorithms with SFO complexity independent of \( n \). Such algorithm is referred as online (streaming) algorithm. For this reason, we propose the online faster SQN method to solve the online problem

\[
\min_{x \in \mathbb{R}^d} f(x) := \mathbb{E}_{\mu \rightarrow \mathcal{P}}[f_\mu(x)]
\]

where \( \mathbb{E}_{\mu \rightarrow \mathcal{P}}[f_\mu(x)] \) denotes a population risk over an underlying data distribution \( \mathcal{P} \). Since the problem can be perceived as having infinite samples, it is impossible to evaluate the full gradient \( \nabla f(x) \) by running across the whole data set. The stochastic sampling thus is adopted as a surrogate strategy. Algorithm 4 shows the detail steps of the proposed online SpiderSQN algorithm.

At steps 3 and 5, the gradient is estimated over the mini-batch samples drawn from the underlying distribution \( \mathcal{P} \). Especially, due to the nature of the online data flow, these samples are sampled without replacement. The variant with vanilla momentum scheme is shown in Algorithm 5. As for the counterparts with epochwise-restart momentum and epochwise-diminishing momentum, one just replaces the \( \alpha_k \) in Algorithm 5 with the one defined in (8) and (9), respectively.

V. CONVERGENCE ANALYSIS

In this section, we analyze the convergence rate of the faster SQN method and its online version. Detailed convergence analysis can be found in [38].

A. Convergence Analysis of Faster SQN Method

First, the convergence properties of the four SpiderSQN-type of algorithms are presented. Let Assumptions 1 to 5 hold, and the following theorems are obtained.

**Theorem 1:** Apply Algorithm 2 to solve the problem (1), and suppose \( x_* \) is its output. Let \( q = |z_k| \equiv \sqrt{n} \), and \( \eta = ((1 + \sqrt{5})\sigma_{\min}/2L\sigma_{\max}^2) \). Then, there is \( x_* \) satisfies

\[
\mathbb{E}\|\nabla f(x_*)\| \leq \epsilon
\]

for any \( \epsilon > 0 \) provided that the iterations number \( K \) satisfies

\[
K \geq \mathcal{O}(\frac{f(x_0) - f^*}{\epsilon^2}).
\]

Moreover, the total number of SFO calls is at most in the order of \( \mathcal{O}(n + n^{1/3} \epsilon^{-2}) \).
Algorithm 5 SpiderSQN-M for Online Nonconvex Optimization

Input: $|\zeta_0|, |\zeta_k|, q, K \in \mathbb{N}, \{\beta_k\}_{k=0}^{K-1} > 0$.
1: Set $a_k = \frac{2}{k+1}$ for $k = 0, \ldots, K$ and $\lambda_k \in [\beta_k, (1 + a_k)\beta_k]$ for $k = 0, \ldots, K - 1$.
2: Initialize $y_0 = x_0 \in \mathbb{R}^d$.
3: for $k = 0, 1, \ldots, K - 1$ do
4: $z_k = (1 - a_k + 1)y_k + a_{k+1}x_k$,
5: if mod$(k, q) = 0$ then
6: Draw $|\zeta_0|$ samples, and compute $v_k = \nabla f_{\theta_k}(z_k)$,
7: else
8: Draw $|\zeta_k|$ samples, and compute $v_k = \nabla f_{\theta_k}(z_k) - \nabla f_{\theta_k}(z_{k-1}) + u_{k-1}$,
9: end if
10: Compute $d_k = H_k v_k$ through SdLBFGS [24],
11: $x_{k+1} = x_k - \lambda_k d_k$,
12: $y_{k+1} = z_k - \beta_k d_k$,
13: end for
14: Output (in theory): $x_\xi$, where $\xi \leq \max \{1, \ldots, K\}$.
15: Output (in practice): $x_K$.

Theorem 2: Apply Algorithm 3 to solve the problem (1), and suppose $z_\xi$ is its output. Let $a_k = (2/(k + 1)), q = |\zeta_\xi| \equiv \sqrt{n}$, $\beta_k \equiv (\sigma_{\text{min}}/3 + \sqrt{\xi})L \sigma_{\text{max}}^2$ and $\lambda_k \in [\beta_k, (1 + a_k)\beta_k]$. Then, there is $z_\xi$ satisfies $\mathbb{E}\{\nabla f(z_\xi)\} \leq \epsilon$ for any $\epsilon > 0$ provided that the iterations number $K$ satisfies

$$K \geq O\left(\frac{f(x_0) - f^*}{\epsilon^2}\right),$$

(12)

Moreover, the total number of SFO calls is at most in the order of $O(n + n^{1/2}\epsilon^{-2})$.

Theorem 3: Apply the SpiderSQN with either epochwise-restart momentum (SpiderSQNMR) or epochwise-diminishing momentum (SpiderSQNMD) to solve the problem (1), and suppose $z_\xi$ is its output. Let $a_k$ defined as (8) and (9) for SpiderSQNM and SpiderSQNMD, respectively. Set $q = |\zeta_\xi| \equiv \sqrt{n}$, $\beta_k \equiv (\sigma_{\text{min}}/(3 + \sqrt{\xi})L \sigma_{\text{max}}^2)$, and $\lambda_k \in [\beta_k, (1 + a_k)\beta_k]$. Then, for both algorithms, there is $x_\xi$ satisfies $\mathbb{E}\{\nabla f(x_\xi)\} \leq \epsilon$ for any $\epsilon > 0$ provided that the iterations number $K$ satisfies

$$K \geq O\left(\frac{f(x_0) - f^*}{\epsilon^2}\right).$$

(13)

Moreover, the total number of SFO calls is at most in the order of $O(n + n^{1/2}\epsilon^{-2})$.

Remark 1: There are two differences between Theorem 1 and Theorem 2&3: 1) Theorem 2&3 introduce an extra parameter, i.e., $a_k$, because of using momentum scheme and 2) the choice of $\beta_k$ in Theorem 2&3 are different from that of $\eta$ in Theorem 1 (note that $\beta_k$ plays the same role as $\eta$). Theorem 2&3 are the same except for the choice of $a_k$ due to using different momentum schemes. Moreover, given required conditions in Theorems 1–3, the SFO complexity of Algorithm 2 and its variants with different momentum schemes to satisfy the $\epsilon$-first-order stationary condition are $O(n + n^{1/2}\epsilon^{-2})$, which matches the state-of-the-art results of first-order stochastic methods.

B. Convergence Analysis of Online Faster SQN Method

To study the SFO complexity of the online SpiderSQN-type of algorithms, we let Assumptions 1–5 hold and make an extra standard assumption (Assumption 6).

Assumption 6: There exists a constant $\sigma_1 > 0$ such that for all $x \in \mathbb{R}^d$ and all random samples $u \sim P$, it holds that $E_{u \sim P}\|f_{\theta_u}(x) - f(x)\|^2 \leq \sigma_1^2$.

Assumption 6 shows that the $f_{\theta_u}(x)$ is an unbiased estimator of $f(x)$ with bounded variance. Assumption 6 is a standard assumption in online optimization analysis [39] and is for online case only.

Theorem 4: Let additional Assumption 6 hold. Apply Algorithm 4 to solve the online optimization problem (10). Choose any desired accuracy $\epsilon > 0$ and set parameters as

$$q = |\zeta_k| = \sqrt{|\zeta_0|} \equiv \sqrt{\left(\frac{\eta \sigma_{\text{max}}}{\beta^*} + 2 + \frac{L^2 \eta^2 \sigma_{\text{max}}^3}{\beta^*}\right)2\sigma_1^2},$$

(14)

where $\beta^* = (\eta \sigma_{\text{min}}/2) - (L \eta^2 \sigma_{\text{max}}^2/2) - (\sigma_1^3 L \sigma_{\text{max}}^2/2)$, and let $\eta \equiv (1 + \sqrt{5})\sigma_{\text{min}}/2L \sigma_{\text{max}}$. Then, the output $x_\xi$ of this algorithm satisfies $\mathbb{E}\{\nabla f(x_\xi)\} \leq \epsilon$ given that the total number of iterations $K$ satisfies

$$K \geq O\left(\frac{f(x_0) - f^*}{\epsilon^2}\right).$$

(15)

Moreover, the SFO complexity is in the order of $O(\epsilon^{-3})$.

Theorem 5: Let additional Assumption 6 hold. Apply online Algorithm 5 to solve the online optimization problem (10). Choose any desired accuracy $\epsilon > 0$ and set parameters as

$$a_k = \frac{2}{k + 1}, \quad q = |\zeta_k| = \sqrt{|\zeta_0|} \equiv \sqrt{\frac{4(1 + \beta/\beta^*)\sigma_1^2}{\epsilon^2}},$$

(16)

where $\beta^* = \beta((\sigma_{\text{min}}/2) - 3L\beta \sigma_{\text{max}}^2/2 - 3L^2 \beta^2 \sigma_{\text{max}}^3)$, $\beta \equiv (\sigma_{\text{min}}/(3 + \sqrt{15})L \sigma_{\text{max}}^2)$. Let $\beta_k = \beta, \lambda_k \in [\beta_k, (1 + a_k)\beta_k]$. Then, the output $x_{\xi}$ of both algorithms satisfy $\mathbb{E}\{\nabla f(x_{\xi})\} \leq \epsilon$ provided that the total number of iterations $K$ satisfies

$$K \geq O\left(\frac{f(x_0) - f^*}{\epsilon^2}\right).$$

(17)

Moreover, the SFO complexity is in the order of $O(\epsilon^{-3})$. 

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.
Remark 2: There are two differences between Theorem 4 and Theorem 5&6: 1) Theorem 5&6 introduce an extra parameter, i.e., momentum coefficient $\alpha k$ because of using momentum scheme; 2) the choice of $\beta k$ in Theorem 5&6 are different from that of $\eta$ in Theorem 4 (note that $\beta k$ plays the same role as $\eta$). Theorem 5 and Theorem 6 are the same except for the choice of $\alpha k$ due to using different momentum schemes. Moreover, given required conditions in Theorems 4 to 6, the SFO complexity of Algorithm 4 and its variants with different momentum schemes to satisfy the $\epsilon$-first-order stationary condition are $O(\epsilon^{-3})$, which matches the state-of-the-art results of first-order stochastic methods.

C. Lower Bound

We will present the optimality of our algorithms in the perspective of algorithmic lower bound result [40], which can be obtained by following the analyses in [9]. For the finite-sum case, given any random algorithm $\mathcal{A}$ that maps functions $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to a sequence of iterates in $\mathbb{R}^{d+1}$, with

$$[x^k; i_k] = \mathcal{A}^{k-1}((\xi, \nabla f_{i_0}(x^0), \nabla f_{i_1}(x^1), \ldots, \nabla f_{i_{k-1}}(x^{k-1}))),$$

$k \geq 1$ (17)

where $\mathcal{A}^k$ denotes measure mapping into $\mathbb{R}^{d+1}$, $i_k$ is the individual function chosen by $\mathcal{A}$ at iteration $k$, and $\xi$ is uniform random vector from $[0, 1]$. Moreover, there is $[x^0; i_0] = A^0(\xi)$, where $A^0$ is a measure mapping. The lower bound result for solving (1) is stated in Theorem 7.

Theorem 7 (Lower Bound for SFO Complexity for the Finite-Sum Case) [9]: For any $L > 0$, $\Delta > 0$, and $2 \leq n \leq O(\Delta^2 L^2 \cdot \epsilon^{-4})$, for any algorithm $\mathcal{A}$ satisfying (17), there exists a dimension $d = O(\Delta^2 L^2 \cdot n^2 \epsilon^{-4})$, and a function $f$ satisfying Assumptions 1–6 for the finite-sum case, such that

in order to find an $\epsilon$-first-order stationary point must cost at least $O(L \Delta \cdot n^{1/2} \epsilon^{-2})$ stochastic gradient accesses.

Note that the condition $n \leq O(\epsilon^{-4})$ in Theorem 7 ensures the lower bound $O(n^{1/2} \epsilon^{-2}) = O(n^{1/2} \epsilon^{-2})$. Therefore, the upper bound in Theorem 1 matches the lower bound in Theorem 7 up to a constant factor of relevant parameters, and is, thus, near-optimal. The proof of Theorem 7 provided in the arXiv version [38] utilizes a specific counterexample function that requires at least $O(n^{1/2} \epsilon^{-2})$ stochastic gradient accesses, which is inspired by [9], [40], and [41].

Remark 3: Through setting $n = O(\epsilon^{-4})$ the lower bound complexity in Theorem 7 can achieve $O(\epsilon^{-3})$. It is necessary to emphasize that this does not violate the upper bound in the online case, i.e., $O(\epsilon^{-3})$ (Theorems 4–6), since the counterexample established in the lower bound depends not on the stochastic gradient variance $\sigma^2$ specified in Assumption 6 but the example number $n$. To obtain the lower bound result for the online case with the additional Assumption 6, one can just construct a counterexample that requires $O(\epsilon^{-3})$ stochastic gradient accesses with the knowledge of $\sigma^2$ instead of $n$.

D. Computational Complexity

In the following, we will analyze the time complexity of the proposed algorithms and show that the extra computation costs of computing the inverse Hessian approximation matrix and using momentum acceleration are negligible.

First, we analyze the computational cost of Algorithm 1. In step 1, the computation of $g_{i_k}^{-1}$ involves two inner product, which takes $2d$ multiplications. In step 2, the computation involves two inner products and one scalar-vector product, which takes $3d$ multiplications. The first recursive loop (i.e., steps 3–5) involves $2m$ scalar-vector multiplications and $m$ vector inner products, which takes $3md$ multiplications.
TABLE II
TOTAL COMPUTATIONAL COMPLEXITIES OF ALGORITHMS 1 TO 5 IN AN OUTER LOOP. ESPECIALLY, AN OUTER LOOP OF ALGORITHMS 2 TO 5 INCLUDES $q$ COMPUTATIONS OF THE STOCHASTIC GRADIENT, $q$ CALLS OF ALGORITHM 1, AND ONE COMPUTATION OF THE FULL GRADIENT

<table>
<thead>
<tr>
<th>Algorithm 1 ($q$ iterations)</th>
<th>Algorithm 2</th>
<th>Algorithm 3</th>
<th>Algorithm 4</th>
<th>Algorithm 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>step complexity</td>
<td>step complexity</td>
<td>step complexity</td>
<td>step complexity</td>
<td>step complexity</td>
</tr>
<tr>
<td>1 $O(d)$</td>
<td>3 $O(nd)$</td>
<td>4 $O(d)$</td>
<td>3 $O(e^{-d})$</td>
<td>4 $O(d)$</td>
</tr>
<tr>
<td>2 $O(d)$</td>
<td>5 $O(n^{1/2}d)$</td>
<td>6 $O(nd)$</td>
<td>5 $O(e^{-1}d)$</td>
<td>6 $O(e^{-2}d)$</td>
</tr>
<tr>
<td>3-6 $O(md)$</td>
<td>7 $O(md)$</td>
<td>8 $O(n^{1/2}d)$</td>
<td>7 $O(md)$</td>
<td>8 $O(e^{-1}d)$</td>
</tr>
<tr>
<td>7 $O(d)$</td>
<td>8 $O(d)$</td>
<td>10 $O(md)$</td>
<td>8 $O(d)$</td>
<td>10 $O(md)$</td>
</tr>
<tr>
<td>8-11 $O(md)$</td>
<td>–</td>
<td>11-12</td>
<td>–</td>
<td>11-12</td>
</tr>
<tr>
<td>total $O(qmd)$</td>
<td>total $O(nd + qmd)$</td>
<td>total $O(nd + qmd)$</td>
<td>total $O(e^{-2}d + qmd)$</td>
<td>total $O(e^{-2}d + qmd)$</td>
</tr>
</tbody>
</table>

So does the second loop (i.e., Steps 8–10). Step 7 involving a scalar-vector product takes $d$ multiplications. Therefore, the whole procedure takes $(6m + 6)d$ multiplications.

Then, we turn to Algorithm 3. Step 4 involves scalar-vector products, which takes $2d$ multiplications. In Step 6, the computation of the full gradient takes at least $2nd$ multiplications. In step 8, the computation of stochastic gradients has batch-size $n^{1/2}$ takes $2n^{1/2}d$ multiplications. In steps 10, $(6m + 6)d$ multiplications are necessary for calling Algorithm 1. Steps 11 and 12 involving scalar-vector products need $d$ multiplications. Therefore, the total computational cost in an outer loop involves $[(6m + 6)q + 2n + 2n^{1/2}q + 4q]d$ multiplications.

Based on the above-mentioned analyses, the computational cost of other algorithms can be obtained easily. For algorithms without momentum acceleration, one needs to omit the extra computation cost ($2d$ multiplications) of computing momentum term. As for algorithms without using approximate Hessian information, one needs to omit the extra computational cost of calling Algorithm 1.

We summarize the computational complexity of each algorithm during an outer loop with $q$ iterations (for finite-sum case, there is $O(q) = O(n^{1/2})$, while for online case, there is $O(q) = O(e^{-t})$) in Table II. As shown in Table II, for finite-sum case, the extra computation costs of the computing approximate Hessian information and using momentum acceleration take up $(mn^{1/2} / (n + mn^{1/2}))$ in the whole procedure. Since $m$ usually ranges from 5 to 20 as suggested in [34] and $n$ is sufficiently large in big data situation, the extra computation thus is negligible. So does the online case, when $\epsilon$ is considerably small. Note that for analysis convenience, we reasonably assume $2d$ multiplications are needed when computing a stochastic gradient for a general machine learning problem.

VI. EXPERIMENTS

In this section, to demonstrate the promising performance of the proposed algorithms, we compare our methods with some state-of-the-art SQN algorithms and stochastic first-order algorithms for nonconvex optimization. Following are brief introductions of algorithms used in our experiments.

**SpiderBoost** [42]: SpiderBoost is a boosting version of SPIDER, which takes up a more aggressive stepsize than SPIDER, and thus, outperforms SPIDER in practice.

**SdLBFGSVR** [24]: SdLBFGSVR is an SQN method (more specifically, SdLBFGS method) equipped with the SVRG variance reduction technique.

**SpiderMED** [39]: ProxSPIDER-MED [39] is a proximal method that uses the epochwise-diminishing momentum scheme to improve the practical performance of SpiderBoost. Especially, ProxSPIDER-MED is the faster one among all momentum variants of SpiderBoost proposed in [39]. Since our article does not touch upon nonconvex nonsmooth optimization, we adopt the ProxSPIDER-MED without a proximal operator and call it SpiderMED.

**Our methods**: Our methods include four SpidersQN (SSQN) type of methods, i.e., SSQN (Algorithm 2), SSQN with vanilla momentum scheme (SSQNM, i.e., Algorithm 3), SSQN with epochwise-restart momentum (SSQNMRM), and SSQN with epochwise-diminishing momentum (SSQNMDM). Note that SSQNMRM and SSQNMDM are proposed in section III-C.

Follow the experiment setting in [39], we choose a fixed mini-batch size 256 and the epoch length $q$ is set to $2n/256$. When implement the SdLBFGS [24], we set the memory size to $m = 5$ as suggested in [34] and fix the $\sigma$ for each comparison. Moreover, we implement experiments on synthetic data for the complement of real data sets, which are generated as [24].

**Generating Synthetic Data**: The training and testing points $(a, b)$ are generated in the following manner. First, we generate a sparse vector $a$ with 5% nonzero components following the uniform distribution on $[0, 1]^n$, and then set $b = \text{sign}(u, a)$ for some $u \in \mathbb{R}^n$ drawn from the uniform distribution on $[-1, 1]^n$.

**Descriptions of Data Sets**: We implement all experiments on five public data sets from the LIBSVM [43] and synthetic data as the complement to these public data sets are summarized in Table III. Especially, as for the mnist data set, we use the one-versus-rest technique to convert it to binary class data.

**A. Nonconvex Support Vector Machine**

First, the above-mentioned algorithms are applied to solve the nonconvex support vector machine (SVM) problem with a sigmoid loss function

$$\min_{x \in \mathbb{R}^p} f(x) := \frac{1}{n} \sum_{i=1}^{n} (1 - \tanh(b_i \langle x, a_i \rangle)) + r\|x\|^2$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.
TABLE III

<table>
<thead>
<tr>
<th>datasets</th>
<th>#samples</th>
<th>#features</th>
<th>#classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a9a</td>
<td>32,561</td>
<td>123</td>
<td>2</td>
</tr>
<tr>
<td>w8a</td>
<td>64,700</td>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>icjnn1</td>
<td>141,691</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>mnist</td>
<td>60,000</td>
<td>780</td>
<td>2</td>
</tr>
<tr>
<td>covtype</td>
<td>581,012</td>
<td>54</td>
<td>2</td>
</tr>
<tr>
<td>synthetic data</td>
<td>100,000</td>
<td>5,000</td>
<td>2</td>
</tr>
</tbody>
</table>

where \( a_i \in \mathbb{R}^d \) denotes the \( i \)th sample and \( b_i \in \pm 1 \) is the corresponding label. In the experiments, the learning rate \( \eta \) and regular coefficient \( r \) for all algorithms are both fixed as 0.001. Moreover, in algorithms with momentum scheme \( \beta_k \) is fixed as \( \eta \), and \( \lambda_k \) remains the same for each comparison.

The experiment results on those four data sets are shown in Fig. 1, where \( f(x) \) is the function value and \( f(x^*) \) is a suitable constant for each case. First, as for w8a and icjnn1 data sets, the initial solutions to all algorithms are drawn from the standard norm distribution, while for the a9a and mnist data sets they take the original point. As Fig. 1 shows, all these SQN methods [including SdlBFGSR and four SpiderSQN (SSQN)-type of algorithms] outperform stochastic first-order methods (including Spider and SpiderMED) by a considerably large margin, which demonstrates the promising nature of SQN methods for nonconvex optimization.

The learning curves on the gap between \( f(x) \) and \( f(x^*) \) are reported in Fig. 2. As one can see from Fig. 2, the SQN methods still have a significantly better performance than the stochastic first-order methods. In addition, the proposed four SSQN-type algorithms outperform the SdlBFGSR with a considerably large margin. In most cases, SSQNMED outperforms SSQNM and SSQNMER by a large gap, except in the mnist data set, where SSQNMER and SSQNMED have similar performances and are both significantly better than SSQNM.

B. Nonconvex Robust Linear Regression

We consider comparing these algorithms for solving such a nonconvex robust linear regression problem

\[
\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} \ell(b_i - \langle x, a_i \rangle)
\]

where the nonconvex loss function is defined as \( \ell(x) := \log(x^2/2 + 1) \). The experiment settings are the same as those in the nonconvex SVM problem, except that the initial solutions in all cases are drawn from the standard norm distribution. The learning curves on the gap between \( f(x) \) and \( f(x^*) \) are reported in Fig. 2. As one can see from Fig. 2, the SQN methods still have a significantly better performance than the stochastic first-order methods. In addition, the proposed four SSQN-type algorithms outperform the SdlBFGSR with a considerably large margin. In most cases, SSQNMED outperforms SSQNM and SSQNMER by a large gap, except in the mnist data set, where SSQNMER and SSQNMED have similar performances and are both significantly better than SSQNM.

C. Nonconvex Logistic Regression

Comparisons are conducted among all algorithms for solving a nonconvex logistic regression problem

\[
\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} \ell(b_i, \langle x, a_i \rangle) + r \sum_{i=1}^{d} \frac{x_i^2}{1 + x_i^2}
\]
where the loss function $\ell$ is set to be the cross-entropy loss. For this problem, the initial solutions to all algorithms on the w8a and a9a are data sets drawn from the standard norm distribution, while experiments on the icnn1 and mnist data sets take the original point. Other experiment settings are the same as those of the nonconvex SVM problem. The learning curves on the gap between $f(x)$ and $f(x^*)$ are reported in Fig. 3. Obviously, the SQN methods outperform those stochastic first-order methods by a significantly large gap. Meanwhile, the proposed four SSQN-type of algorithms all have a better performance than the SdLBFGSVR. As for the four SSQN-type of algorithms, their performance is related to the momentum coefficient setting, which means that algorithms with a larger momentum coefficient will converge faster. Moreover, in all cases, the SSQNMEED has the best performance among four SSQN-type algorithms and SSQN has the worst.

**VII. CONCLUSION**

In this article, we presented the novel faster SQN (SpiderSQN) methods. Moreover, we proved that the SpiderSQN methods reach the best known SFO complexity of $O(\min(n + n^{1/2}\epsilon^{-2}, \epsilon^{-3}))$ for finding an $\epsilon$-approximated stationary point. At the same time, we studied the lower bound of SFO complexity of the SpiderSQN methods. As presented in the theoretical results, our methods reach the near-optimal SFO complexity in solving the nonconvex problems. Moreover, we applied three different momentum schemes to SpiderSQN to further improve its practical performance.

**ACKNOWLEDGMENT**

The authors would like to thank the anonymous reviewers for their helpful comments. They would also like to thank the IT Help Desk at the University of Pittsburgh, Pittsburgh, PA, USA.

**REFERENCES**


This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Qingsong Zhang received the B.E. degree in electronic and information engineering from Xidian University, Xi’an, China, in 2018, where he is currently pursuing the Ph.D. degree with the School of Electronic Engineering. His current research interests include machine learning, optimization, and federated learning.

Feihu Huang received the B.S. degree in computational mathematics from Nanchang Hangkong University, Nanchang, China, in 2009, and the M.S. degree in computational mathematics and the Ph.D. degree in computer science and technology from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2013 and 2017, respectively. He is currently a Post-Doctoral Researcher with the University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include machine learning and optimization.

Cheng Deng (Senior Member, IEEE) received the B.E., M.S., and Ph.D. degrees in signal and information processing from Xidian University, Xi’an, China, in 2001, 2006, and 2009, respectively. He is currently a Full Professor with the School of Electronic Engineering, Xidian University. His research interests include computer vision, pattern recognition, and information hiding. He has authored and coauthored more than 80 scientific articles at top venues, including the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS), the IEEE TRANSACTIONS ON IMAGE PROCESSING (TIP), the IEEE TRANSACTIONS ON CYBERNETICS (TCYB), the IEEE TRANSACTIONS ON MULTIMEDIA (TMM), the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS (TCSMC), the IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), the International Conference on Machine Learning (ICML), Neural Information Processing Systems (NIPS), the International Joint Conference on Artificial Intelligence (IJCAI), and the AAAI (Association for the Advancement of Artificial Intelligence) Conference on Artificial Intelligence (AAAI).

Heng Huang received the B.S. and M.S. degrees from Shanghai Jiao Tong University, Shanghai, China, in 1997 and 2001, respectively, and the Ph.D. degree in computer science from the Dartmouth College, Hanover, NH, USA, in 2006. He is currently the John A. Jurenko Endowed Professor in computer engineering with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA. He is also a Consulting Researcher with JD Finance America Corporation, Mountain View, CA, USA. His research interests include machine learning, data mining, computer vision, and biomedical data science.