A smoothing stochastic gradient method for composite optimization

Qihang Lin\textsuperscript{a}, Xi Chen\textsuperscript{b} and Javier Peña\textsuperscript{c}*

\textsuperscript{a}Tippie College of Business, The University of Iowa, Iowa City, IA, 52245, USA; \textsuperscript{b}Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, 94720; \textsuperscript{c}Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA

(Received 28 February 2013; accepted 3 February 2014)

We consider the unconstrained optimization problem whose objective function is composed of a smooth and a non-smooth components where the smooth component is the expectation of a random function. This type of problem arises in some interesting applications in machine learning. We propose a stochastic gradient descent algorithm for this class of optimization problems. When the non-smooth component has a particular structure, we propose a stochastic gradient descent algorithm by incorporating a smoothing method into our first algorithm. The proofs of the convergence rates of these two algorithms are given and we show the numerical performance of our algorithm by applying them to regularized linear regression and logistic regression problems with different sets of synthetic data.

Keywords: first-order method; stochastic gradient; smoothing

1. Introduction

In the past decade, convex programming has been widely applied in a variety of areas including statistical estimation, machine learning, data mining and signal processing. One of the most popular classes of convex programming problems, which appears in many different applications such as Lasso [23] and group Lasso [28], can be formulated as the following minimization problem,

\[ \min_x \phi(x) = f(x) + h(x). \]  

(1)

Here, the function \( f(x) \) is smooth and convex and its gradient \( \nabla f(x) \) is Lipschitz continuous with a Lipschitz constant \( L \). The function \( h(x) \) is assumed to be convex but non-smooth.

Interior-point methods [3,14] are considered as general algorithms for solving different types of convex programming. However, they are not scalable for problems with even moderate sizes due to the big cost of solving the Newton system of linear equations at each main iteration. A block coordinate method was developed by Tseng and Yun [27] and applied to the problems which can be formulated by (1) in [5,15]. However, this method requires a separable structure in the objective function that does not exist in some applications such as overlapped group Lasso [10].

Recently, gradient descent methods, or so called first-order methods, e.g. [1,2,18–20,26], have attracted great interest because they are not only relatively easy to implement but also capable
of solving some challenging problems with huge size. For problems formulated as in (1), the first-order methods proposed in [2,18,26] can achieve a $O(1/T^2)$ convergence rate, where $T$ is the number of iterations. The different variations of gradient descent algorithm have been successively applied to different types of problems, for example, nuclear norm regularization [22,25], $\ell_1/\ell_2$-norm regularization [13] and so on.

In each loop of a gradient descent algorithm, a proximal mapping, which itself is a minimization problem, must be solved in order to find the next intermediate solution. Although a proximal mapping usually has a closed form solution which guarantees the efficiency of a gradient descent algorithm, there exists a class of problems in machine learning formulated by (1) including overlapped group Lasso [10] and fused Lasso [24], for which a closed form solution for the proximal mapping is not available. Fortunately, for these problems (e.g. overlapping group Lasso or fused Lasso), the non-smooth component $h(x)$ in the objective function has a particular type of structure of the form,

$$h(x) = \max_{v \in Q} v^T A x,$$

where $Q$ is a compact set. Nesterov [19] proposed a scheme to construct a smooth approximation of the objective function $\phi(x)$ in (1) whenever $h(x)$ satisfies (2) and also a gradient descent algorithm to minimize the approximated problem. This approximation scheme and algorithm have been applied to overlapped group Lasso and fused Lasso [4] and given good numerical results.

A stochastic gradient descent algorithm can be considered as a gradient descent algorithm that utilizes random approximations of gradients instead of exact gradients. During the past few years, a significant amount of work has been done to develop stochastic gradient descent algorithms for different problems (see, e.g. [6,8,11,17]). One reason for people to consider stochastic gradient is that the exact gradients are computationally expensive or sometimes even impossible to evaluate. One typical case to consider stochastic gradients is stochastic optimization where the objective function $f(x)$ is given as an expectation of a random value function $F(x, \xi)$, i.e., $f(x) = E F(x, \xi)$, where $\xi$ is a random variable and the expectation is taken over $\xi$. In this situation, a multidimensional numerical integral would be needed to compute the exact gradient $\nabla f(x) = E \nabla F(x, \xi)$.

That would be too time consuming especially when the dimension is high. It could be even worse if the distribution of $\xi$ is unknown so that there is no way to get the exact gradient $\nabla f(x)$.

In this paper, we consider the optimization problem formulated by (1) but we further assume the smooth component of the objective function to be of the form $f(x) = E F(x, \xi)$ just as it is the case in a stochastic optimization problem. Hence, we have to consider the stochastic gradient for the reasons we mentioned above. We propose a stochastic gradient descent algorithm to solve (1) under the assumption that a stochastic approximation of $\nabla f(x)$, denoted by $G(x, \xi)$, is available in each iteration, where $\xi$ is a random variable. We also assume $G(x, \xi)$ is an unbiased estimate of $\nabla f(x)$, i.e. $E G(x, \xi) = \nabla f(x)$, and $E \| \nabla f(x) - G(x, \xi) \|^2 \leq \sigma^2$ for some non-negative constant $\sigma$. We show that our stochastic gradient algorithm obtain a convergence rate of $O(1/\sqrt{T})$, which is the same, up to a constant independent of $T$, as the convergence rates showed in [6,8,11,17] without assuming strongly convexity for the objective functions.

Similar to the exact gradient descent algorithm, the existing stochastic gradient descent algorithms, e.g. [6,8], have to solve a proximal mapping in each iteration, which may not have a closed form solution. However, when the function $h(x)$ in (1) has the structure given in (2), we propose another stochastic gradient descent algorithm by incorporating the smoothing technique proposed by Nesterov [19]. This method replaces $h(x)$ by its smooth approximation such that the proximal mapping always obtains a closed form solution. Hence, our method can be applied to problems like overlapped group Lasso and fused Lasso, which other stochastic gradient descent algorithms can not solve efficiently.
According to [12,19], the convergence rates of the accelerated gradient algorithms will be reduced from $O(1/T^2)$ to $O(1/T)$ if the smoothing technique in [19] is applied. However, we show that the convergence rate for our stochastic gradient algorithm remains $O(1/\sqrt{T})$ even when the smoothing technique is applied. In other words, although the price of the smoothing technique is high for deterministic gradient methods, it is totally free for stochastic gradient methods.

The rest of this paper is organized as follows: in the next section, we discuss the some related work on stochastic gradient methods. In Section 3, we present our stochastic gradient descent algorithm and prove its convergence rate. Combining the first algorithm with a smoothing technique, we propose another stochastic gradient descent algorithm in Section 4. In Section 5, we show the numerical results on simulated data, followed by a concluding section.

2. Related work

Many deterministic gradient methods have been developed in the literature, e.g. [1,2,18–20,26]. Different from all of these works which require knowing the exact gradient of the objective function, our algorithms can be applied to the optimization problems where only stochastic gradient, which is a noisy estimation of the exact gradient, is available. Our algorithms can be viewed as an extension of the Algorithm 1 in [26] by utilizing stochastic gradients. However, simply replacing the deterministic gradient in [26, Algorithm 1] by a stochastic one will compromise its convergence. To address this challenge, instead of using only one sequence of step lengths throughout the entire algorithm ($\theta_t$ in [26, Algorithm 1]), we introduce an extra sequence of step lengths, denoted by $\gamma_t$, for the proximal mapping step (see Algorithm 1 in the next section for more details). With the proper choice of both $\theta_t$ and $\gamma_t$, our algorithm achieves the asymptotically optimal convergence rate.

The choice of the new step length sequence $\gamma_t$ is inspired by the choice of a similar parameter in the accelerated stochastic approximation (AC-SA) method [11] by Lan. Different from AC-SA where $G(x, \xi)$ represents a stochastic subgradient for the whole non-smooth objective function $\phi(x) = f(x) + h(x)$ in (1), we assume $G(x, \xi)$ is a stochastic gradient only for the smooth component $f(x)$ in (1) and approximate $h(x)$ using the smoothing technique [19]. Ghadimi and Lan proposed another variant of AC-SA algorithm in [6] which uses stochastic gradient of $f(x)$ instead of the stochastic subgradient of $\phi(x)$. However, this algorithm requires $h(x)$ to be simple enough, e.g. $\ell_1$-norm of $x$, so that it can not be applied to a problem with a sophisticated $h(x)$ such as overlapped group Lasso [10] and fused Lasso [24], which the method in this paper can solve efficiently. Moreover, the convergence result established in [6] requires knowing the noise level $\sigma$ of the stochastic gradient, which is hard to estimate in practice. By contrast, our methods do not require this information.

3. Stochastic gradient descent algorithm

In this section, we propose a stochastic gradient algorithm summarized in Algorithm 1 below to solve the following optimization problem:

$$\min_x \phi(x) \equiv f(x) + h(x) = EF(x, \xi) + h(x),$$

where the expectation $E$ is taken over the random variable $\xi$. We assume that at every point $x$, there is a random vector $G(x, \xi)$ determined by $x$ and $\xi$ such that $E G(x, \xi) = \nabla f(x)$ and $E \| \nabla f(x) - G(x, \xi) \|^2 \leq \sigma^2$ for any $x$. In the $t$th iteration of Algorithm 1, we independently generate a random
variable $\xi_t$ from the distribution of $\xi$ and compute $G(y_t, \xi_t)$ at a point $y_t$ based on $\xi_t$. The notation $(x, y)$ in step 3 of Algorithm 1 represents the inner product of $x$ and $y$ in $\mathbb{R}^n$, i.e., $(x, y) = \sum_{i=1}^{n} x_i y_i$.

**Algorithm 1** Stochastic gradient descent algorithm (SG)

**Input:** The total number of iterations $T$ and the Lipschitz constant $L$ of $\nabla f(x)$.

**Initialization:** Choose $\theta_t = \frac{1}{T+4}$ and $\gamma_t = \frac{2}{T+2}(\frac{T+1}{L} + 2)$. Set $x_0 = 0$, $z_0 = 0$.

**Iterate** for $t = 0, 1, 2, \ldots, T$:

1. $y_t = (1 - \theta_t)x_t + \theta_t z_t$.
2. Generate a random vector $\xi_t$ independently from the distribution of $\xi$.
3. $z_{t+1} = \arg\min_x \{(x, G(y_t, \xi_t)) + \frac{\gamma_t L}{2}\|x - z_t\|^2 + h(x)\}$.
4. $x_{t+1} = (1 - \theta_t)x_t + \theta_t z_{t+1}$

**Output:** the solution from the $(T + 1)$-th iteration $x_{T+1}$.

**Remark 1** Algorithm 1 is based on the Algorithm 1 proposed by Tseng in [26]. Algorithm 1 is different from Tseng’s algorithm in two aspects. First, the exact gradient used in Tseng’s algorithm is replaced by the stochastic gradient due to the difficulty of computing the exact gradient in our problems as mentioned in Section 1. Second, two sequences of step lengths, $\theta_t$ and $\gamma_t$, are maintained to guarantee the convergence of our algorithm while in Tseng’s algorithm, one sequence of step length $\theta_t$ is enough. It should be pointed out that if we set $\gamma_t$ in Algorithm 1 to be $2\gamma^*/L(t + 1)$ with $\gamma^* = \max\{2L, 2\sigma^2(T + 1)(T + 2)/3\|x_0 - x^*\|^2\}^{1/2}$, Algorithm 1 just becomes the AC-SA algorithm proposed by Ghadimi and Lan [6] for unconstrained optimization problems when $f(x)$ is just convex but not necessarily strongly convex. However, the parameter $\gamma^*$ in the AC-SA algorithm is hard to evaluate because it depends on the optimal solution $x^*$ and $\sigma$ while the parameters in our algorithm are relatively simple and result in better numerical performances as shown in Section 4.

Here, we assume the proximal mapping in Step 3 in Algorithm 1 can be solved efficiently or has a closed form solution. This is true in many problems where the non-smooth term $h(x)$ is $\ell_1$-norm [23], $\ell_1/\ell_2$-norm [9,13,28] or nuclear norm of $x$ [22,25].

Using the same notation as in Algorithm 1, we present the convergence rate of Algorithm 1 in the following theorem. Some techniques in the proof are inspired by the proofs for the complexity results in [11,26].

**Theorem 1** Suppose $T$ is the total number of iterations in Algorithm 1 and $x^*$ is the optimal solution of (3) and we assume that the stochastic gradient $G(x, \xi)$ satisfies $\mathbb{E}\|\nabla f(x) - G(x, \xi)\|^2 \leq \sigma^2$ for all $x$. Then we have

$$\mathbb{E}(\phi(x_{T+1}) - \phi(x^*)) \leq \frac{2D^2 + \sigma^2}{(T + 2)^{1/2}} + \frac{4D^2 + 2\sigma^2}{(T + 2)^2},$$

where $D = \|x^* - z_0\|$.

Three technical lemmas are presented here before the proof of the convergence rate of Algorithm 1 is given. Lemma 1 is an inequality satisfied by the step lengths we choose in Algorithm 1 and Lemma 2 is a basic property of convex functions. Lemma 3, shown in [11], is a technical result used to characterize the optimal solution of the proximal mapping in step 3 of Algorithm 1. We include a proof of Lemma 3 here for completeness.
Lemma 1 Suppose the sequences \( \{ \theta_t \} \) and \( \{ \gamma_t \} \) are chosen as in Algorithm 1, we have \((1 - \theta_{t+1})/\theta_{t+1} \gamma_{t+1} < 1/\gamma_t \) and \( \gamma_t > \theta_t \).

Proof The first inequality comes from
\[
\frac{\theta_t (1 - \theta_{t+1})}{\theta_{t+1}} = \frac{2/(t+2)(1 - 2/(t+3))}{2/(t+3)} = \frac{t+1}{t+2} \leq \frac{t+2}{t+3} = \frac{\gamma_{t+1}}{\gamma_t}.
\]
The second one follows by inspection. ■

Lemma 2 If \( f(x) \) is a smooth convex function on \( \mathbb{R}^n \) and \( \nabla f(x) \) is Lipschitz continuous with a Lipschitz constant \( L \), then we have
\[
f(y) + \langle x - y, \nabla f(y) \rangle \leq f(x) \leq f(y) + \langle x - y, \nabla f(y) \rangle + \frac{L}{2} \| x - y \|^2
\]
for all \( x, y \).

This is a classical property of convex functions. For a proof, see [18, Lemma 1.2.3 on p. 22] or [7, Theorem B.4.1.1 on p. 110].

Lemma 3 (see also [6,11,26]) Suppose \( \psi(x) \) is convex and \( z^* \) is the optimal solution of
\[
\min_z \psi(z) + \frac{1}{2} \| z - \hat{z} \|^2,
\]
then we have the following inequality:
\[
\psi(z^*) + \frac{1}{2} \| z^* - \hat{z} \|^2 \leq \psi(x) + \frac{1}{2} \| x - \hat{z} \|^2 - \frac{1}{2} \| x - z^* \|^2
\]
for all \( x \).

Proof The definition of \( z^* \) implies that there exists a subgradient \( \eta \) in \( \partial \psi(z^*) \), the subdifferential of function \( \psi(z) \) at \( z^* \), such that
\[
\langle \eta + z^* - \hat{z}, x - z^* \rangle \geq 0 \quad \text{for all } x.
\]
(4)

And the convexity of \( \psi(x) \) implies
\[
\psi(x) \geq \psi(z^*) + \langle \eta, x - z^* \rangle \quad \text{for all } x.
\]
(5)

It is easy to verify that
\[
\frac{1}{2} \| \hat{z} - x \|^2 = \frac{1}{2} \| \hat{z} - z^* \|^2 + \langle z^* - \hat{z}, x - z^* \rangle + \frac{1}{2} \| z^* - x \|^2 \quad \text{for all } x.
\]
(6)

Using the (4)–(6) above, we conclude that
\[
\psi(x) + \frac{1}{2} \| x - \hat{z} \|^2 = \psi(x) + \frac{1}{2} \| \hat{z} - z^* \|^2 + \langle z^* - \hat{z}, x - z^* \rangle + \frac{1}{2} \| z^* - x \|^2
\]
\[
\geq \psi(z^*) + \frac{1}{2} \| \hat{z} - z^* \|^2 + \langle \eta + z^* - \hat{z}, x - z^* \rangle + \frac{1}{2} \| z^* - x \|^2
\]
\[
\geq \psi(z^*) + \frac{1}{2} \| \hat{z} - z^* \|^2 + \frac{1}{2} \| z^* - x \|^2 \quad \text{for all } x.
\]
■
Next, we give the proof of Theorem 1.

**Proof.** We define \( \Delta_t = \nabla f(y_t) - G(y_t, \xi_t) \) so that \( \mathbb{E} \Delta_t = 0 \) and \( \mathbb{E} \| \Delta_t \|^2 \leq \sigma^2 \). We can bound \( \phi(x_{t+1}) \) from above as follows.

\[
\phi(x_{t+1}) = f(x_{t+1}) + h(x_{t+1}) \\
\leq f(y_t) + \langle x_{t+1} - y_t, \nabla f(y_t) \rangle + \frac{L}{2} \| x_{t+1} - y_t \|^2 + h(x_{t+1}) \\
\leq (1 - \theta_t) \phi(y_t) + (1 - \theta_t) \| x_t - y_t, \nabla f(y_t) \| + h(x_t) \\
+ \theta_t (f(y_t) + \langle z_{t+1} - y_t, \nabla f(y_t) \rangle + h(z_{t+1})) + \frac{L}{2} \| z_{t+1} - z_t \|^2 \\
\leq (1 - \theta_t) \phi(x_t) + \theta_t (f(y_t) + \langle z_{t+1} - y_t, G(y_t, \xi_t) \rangle + h(z_{t+1}) + \frac{L}{2} \| z_{t+1} - z_t \|^2 \\
+ (\theta_t^2 - \theta_t \gamma_t) \frac{L}{2} \| z_{t+1} - z_t \|^2 + \theta_t \langle z_{t+1} - y_t, \Delta_t \rangle. \tag{7}
\]

The first and third inequalities above are due to Lemma 2 and the second one is implied by the updating equations for \( y_t \) and \( x_{t+1} \) and the convexity of \( h(x) \).

According to Lemma 3 with \( \psi(z) = (1/\gamma_t L)(z, G(y_t, \xi_t)) + h(z) \), \( z^* = z_{t+1} \) and \( \hat{z} = z_t \), we get

\[
(\langle z_{t+1}, G(y_t, \xi_t) \rangle + h(z_{t+1})) + \frac{\gamma_t L}{2} \| z_{t+1} - z_t \|^2 \\
\leq (\langle x, G(y_t, \xi_t) \rangle + h(x)) + \frac{\gamma_t L}{2} \| x - z_t \|^2 - \frac{\gamma_t L}{2} \| x - z_{t+1} \|^2 \quad \text{for all } x. \tag{8}
\]

By choosing \( x = x^* \) in (8), it follows from (7) and (8) that

\[
\phi(x_{t+1}) \leq (1 - \theta_t) \phi(x_t) + \theta_t \left( f(y_t) + \langle x^* - y_t, G(y_t, \xi_t) \rangle + h(x^*) + \frac{L}{2} \| x^* - z_t \|^2 \right) \\
- \theta_t \gamma_t \frac{L}{2} \| x^* - z_{t+1} \|^2 + (\theta_t^2 - \theta_t \gamma_t) \frac{L}{2} \| z_{t+1} - z_t \|^2 + \theta_t \langle z_{t+1} - y_t, \Delta_t \rangle \\
= (1 - \theta_t) \phi(x_t) + \theta_t \left( f(y_t) + \langle x^* - y_t, \nabla f(y_t) \rangle + h(x^*) + \gamma_t \frac{L}{2} \| x^* - z_t \|^2 \right) \\
- \theta_t \gamma_t \frac{L}{2} \| x^* - z_{t+1} \|^2 + (\theta_t^2 - \theta_t \gamma_t) \frac{L}{2} \| z_{t+1} - z_t \|^2 + \theta_t \langle z_{t+1} - x^*, \Delta_t \rangle. \tag{9}
\]

Here, the equality above holds because \( \Delta_t = \nabla f(y_t) - G(y_t, \xi_t) \).
By Lemma 2, the term \( f(y_t) + \langle x^* - y_t, \nabla f(y_t) \rangle + h(x^*) \) in (9) is no more than \( \phi(x^*) \). Hence, we can upper bound \( \phi(x_{t+1}) \) as:

\[
\phi(x_{t+1}) \leq (1 - \theta_t)\phi(x_t) + \theta_t\phi(x^*) + \theta_t\gamma_t \frac{L}{2}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\|x^* - z_{t+1}\|^2 \\
- (\theta_t\gamma_t - \theta_t^2) \frac{L}{2}\|z_{t+1} - z_t\|^2 + \theta_t\langle z_{t+1} - y_t, \Delta_t \rangle - \theta_t\langle x^* - y_t, \Delta_t \rangle \\
= (1 - \theta_t)\phi(x_t) + \theta_t\phi(x^*) + \theta_t\gamma_t \frac{L}{2}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\|x^* - z_{t+1}\|^2 \\
- (\theta_t\gamma_t - \theta_t^2) \frac{L}{2}\|z_{t+1} - z_t\|^2 + \theta_t\langle z_{t+1} - z_t, \Delta_t \rangle + \theta_t\langle z_t - x^*, \Delta_t \rangle \\
\leq (1 - \theta_t)\phi(x_t) + \theta_t\phi(x^*) + \theta_t\gamma_t \frac{L}{2}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\|x^* - z_{t+1}\|^2 \\
- (\theta_t\gamma_t - \theta_t^2) \frac{L}{2}\|z_{t+1} - z_t\|^2 + \theta_t\|z_{t+1} - z_t\|\|\Delta_t\| + \theta_t\langle z_t - x^*, \Delta_t \rangle \\
\leq (1 - \theta_t)\phi(x_t) + \theta_t\phi(x^*) + \theta_t\gamma_t \frac{L}{2}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\|x^* - z_{t+1}\|^2 \\
+ \frac{\theta_t\|\Delta_t\|^2}{2L(\gamma_t - \theta_t)} + \theta_t\langle z_t - x^*, \Delta_t \rangle.
\]

We get the second inequality above by applying Cauchy–Schwarz inequality to \( \langle z_{t+1} - z_t, \Delta_t \rangle \) and the last inequality comes from applying the inequality \(-ax^2 + bx \leq b^2/4a\) with \( a > 0 \) to \( a = (\theta_t\gamma_t - \theta_t^2), x = \|z_{t+1} - z_t\| \) and \( b = \theta_t\|\nabla_t\|. \) Note that \( (\theta_t\gamma_t - \theta_t^2) > 0 \) from Lemma 1.

Until now, we have already got

\[
\phi(x_{t+1}) \leq (1 - \theta_t)\phi(x_t) + \theta_t\phi(x^*) + \theta_t\gamma_t \frac{L}{2}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\|x^* - z_{t+1}\|^2 \\
+ \frac{\theta_t\|\Delta_t\|^2}{2L(\gamma_t - \theta_t)} + \theta_t\langle z_t - x^*, \Delta_t \rangle.
\]

(10)

We define \( \mathbb{E}(\langle z_t - x^*, \Delta_t \rangle | \xi_1, \ldots, \xi_{t-1}) \) to be the conditional expectation of \( \langle z_t - x^*, \Delta_t \rangle \) under the condition that \( \xi_1, \ldots, \xi_{t-1} \) have been generated. According to Algorithm 1, \( z_t \) is only determined by \( \xi_1, \ldots, \xi_{t-1} \) but not by \( \xi_t \). Hence, \( \mathbb{E}(\langle z_t - x^*, \Delta_t \rangle | \xi_1, \ldots, \xi_{t-1}) = 0 \) because \( \mathbb{E}\Delta_t = 0 \). By the iterative property of expectation, we have

\[ \mathbb{E}\langle z_t - x^*, \Delta_t \rangle = \mathbb{E}(\mathbb{E}(\langle z_t - x^*, \Delta_t \rangle | \xi_1, \ldots, \xi_{t-1})) = \mathbb{E}0 = 0. \]

Hence, if we subtract \( \phi(x^*) \) from both sides of inequality (10) and take the expectation, we will have

\[
\mathbb{E}(\phi(x_{t+1}) - \phi(x^*)) \leq (1 - \theta_t)(\mathbb{E}(\phi(x_t)) - \phi(x^*)) \\
+ \theta_t\gamma_t \frac{L}{2}\mathbb{E}\|x^* - z_t\|^2 - \theta_t\gamma_t \frac{L}{2}\mathbb{E}\|x^* - z_{t+1}\|^2 + \frac{\theta_t\sigma^2}{2L(\gamma_t - \theta_t)}.
\]

(11)
Moreover, we divide both sides of inequality (11) by $\theta_t \gamma_t$ and get
\[
\frac{1}{\theta_t \gamma_t} (\mathbb{E}(\phi(x_{t+1})) - \phi(x^*)) \leq \frac{1 - \theta_t}{\theta_t \gamma_t} (\mathbb{E}(\phi(x_t)) - \phi(x^*)) + \frac{L}{2} \mathbb{E}\|x^* - z_t\|^2 - \frac{L}{2} \mathbb{E}\|x^* - z_{t+1}\|^2 + \frac{\sigma^2}{2L \gamma_t (\gamma_t - \theta_t)} \\
\leq \frac{1}{\theta_{t-1} \gamma_{t-1}} (\mathbb{E}(\phi(x_t)) - \phi(x^*)) + \frac{L}{2} \mathbb{E}\|x^* - z_t\|^2 - \frac{L}{2} \mathbb{E}\|x^* - z_{t+1}\|^2 + \frac{\sigma^2}{2L \gamma_t (\gamma_t - \theta_t)}, \tag{12}
\]
where the second inequality comes from Lemma 1.

By applying inequality (12) recursively, we obtain
\[
\frac{1}{\theta_T \gamma_T} (\mathbb{E}(\phi(x_{T+1})) - \phi(x^*)) \leq \frac{L}{2} \mathbb{E}\|x^* - z_0\|^2 + \frac{\sigma^2}{2L} \sum_{t=0}^{T} \frac{1}{\gamma_t (\gamma_t - \theta_t)}. \tag{13}
\]

The definitions of $\theta_t$ and $\gamma_t$ imply
\[
\sum_{t=0}^{T} \frac{1}{\gamma_t (\gamma_t - \theta_t)} = \sum_{t=0}^{T} \frac{(t + 2)^2}{4(T^{3/2}/L + 2)(T^{3/2}/L + 1)} \leq \frac{(T + 2)(T + 3)(2T + 5)L^2}{24T^3} \leq \frac{12T^3 L^2}{24T^3} = \frac{L^2}{2},
\]
which, together with inequality (13), implies:
\[
\mathbb{E}(\phi(x_{T+1})) - \phi(x^*) \leq \theta_T \gamma_T \left( \frac{L}{2} D^2 + \frac{\sigma^2 L}{4} \right) \leq \frac{4}{(T + 2)^2} \left( \frac{T^{3/2}}{L} \left( \frac{L}{2} D^2 + \frac{\sigma^2 L}{4} \right) + \frac{8}{(T + 2)^2} \left( \frac{L}{2} D^2 + \frac{\sigma^2 L}{4} \right) \right) \leq \frac{2D^2 + \sigma^2}{(T + 2)^{3/2}} + L \frac{4D^2 + 2\sigma^2}{(T + 2)^2}.
\]

**Remark 2** Algorithm 1 obtains an asymptotically rate of convergence $\mathbb{E}(\phi(x_{T+1}) - \phi(x^*)) = O(1/\sqrt{T})$ which is the same as the convergence rate of the AC-SA algorithm proposed by Ghadimi and Lan [6] up to a constant factor. This convergence rate is also known to be asymptotically optimal (see [16, Chapter 5, pp. 197–205]) in terms of the number of iterations $T$.

### 4. Smoothing stochastic gradient descent algorithm

Notice that a proximal mapping
\[
z_{t+1} = \arg\min_x \left\{ \langle x, G(y, \xi_t) \rangle + \frac{\gamma_t L}{2} \|x - z_t\|^2 + h(x) \right\} \tag{14}
\]
must be solved in the Step 3 of Algorithm 1. Similar type of proximal mappings also appear in other deterministic or stochastic gradient algorithms such as the ones in [4,6,8,10,13,17,21,22,26].
As indicated in Section 1, (14) does not necessarily have a closed form solution. This happens, in particular, in group Lasso problem with overlapped group structures [10] and fused Lasso [24]. In this case, another iterative algorithm has to be designed for solving this proximal mapping in each iteration of Algorithm 1, which could make Algorithm 1 very slow for practical applications.

In order to modify Algorithm 1 for problems whose corresponding proximal mappings have no closed form, we utilize the smoothing technique proposed by Nesterov [19] to construct a smooth approximation for problem (3) before we apply Algorithm 1.

Suppose the non-smooth part \( h(x) \) in (3) can be represented as

\[
 h(x) = \max_{v \in Q} v^T Ax,
\]

we consider the function

\[
 h_\mu(x) = \max_{v \in Q} \{ v^T Ax - \mu d(v) \}.
\]  \hspace{1cm} (15)

Here, the parameter \( \mu \) is a positive constant and \( d(v) \) is a smooth and strongly convex function on \( Q \). According to Nesterov [19], the function \( h_\mu(x) \) is a smooth lower approximation for \( h(x) \) if \( \mu \) is positive. In fact, it can be shown that

\[
 h_\mu(x) \leq h(x) \leq h_\mu(x) + \mu M \quad \text{for all } x,
\]

where \( M = \max_{v \in Q} d(v) \). Hence, the parameter \( \mu \) controls the accuracy of approximation.

We denote by \( v_\mu(x) \) the optimal solution of the maximization problem involved in (15). Since \( d(v) \) is strongly convex, \( v_\mu(x) \) is well-defined because of the uniqueness of the optimal solution. It is proved in [19] that \( h_\mu(x) \) is a smooth function whose gradient is

\[
 \nabla h_\mu(x) = A^T v_\mu(x). \]  \hspace{1cm} (16)

Therefore, the function \( \phi_\mu(x) \equiv f(x) + h_\mu(x) \) performs as a smooth lower approximation for \( \phi(x) \) in problem (3) and we have

\[
 \phi_\mu(x) \leq \phi(x) \leq \phi_\mu(x) + \mu M \quad \text{for all } x. \]  \hspace{1cm} (17)

By (16), the gradient of \( \phi_\mu(x) \) is

\[
 \nabla \phi_\mu(x) = \nabla f(x) + A^T v_\mu(x) \]  \hspace{1cm} (18)

and \( G(x, \xi) + A^T v_\mu(x) \) provides its stochastic approximation. It is easy to see that \( \mathbb{E}(G(x, \xi) + A^T v_\mu(x)) = \nabla \phi_\mu(x) \) and \( \mathbb{E}\|\nabla \phi_\mu(x) - G(x, \xi) - A^T v_\mu(x)\|^2 \leq \sigma^2 \) for any \( x \).

It is also shown in [19] that the gradient \( \nabla \phi_\mu(x) \) is Lipschitz-continuous with a Lipschitz constant

\[
 L_\mu = L + \frac{1}{c\mu} \|A\|^2, \]  \hspace{1cm} (19)

where \( \|A\| = \max_{\|x\| = 1, \|y\| = 1} y^T Ax \) and \( c > 0 \) is the strong convexity parameter of function \( d(v) \).

Since \( \phi_\mu(x) \) is a smooth function with a stochastic gradient \( G(x, \xi) + A^T v_\mu(x) \) at each \( x \), we can apply Algorithm 1 to minimize \( \phi_\mu(x) \). When the smooth parameter \( \mu \) is small enough, the solution we get will also be a good approximate solution for (3). The choice \( \mu \) depends on the accuracy of the solution the user expect to have. As we are going to show later, in order to achieve a solution with \( \mathbb{E}(\phi(x_{T+1}) - \phi(x^*)) \leq O(1/\sqrt{T}) \) after \( T \) iterations, one should choose \( \mu \) as small as \( O(1/T) \). This modified algorithm is proposed as Algorithm 2 as follows.
Algorithm 2 Smoothing stochastic gradient descent algorithm (SSG)

**Input:** The total number of iterations $T$, the Lipschitz constant $L$ for $\nabla f(x)$ and the smooth parameter $\mu$.

**Initialization:** Compute the Lipschitz constant $L_\mu$ by (19). Choose $\theta_t = \frac{2}{\gamma_t}$ and $\gamma_t = \frac{2}{\mu_t} \left( \frac{T^{3/2}}{L_\mu} + 2 \right)$. Set $x_0 = 0$, $z_0 = 0$ and $t = 0$.

**Iterate** for $t = 0, 1, 2, \ldots T$:

1. $y_t = (1 - \theta_t)x_t + \theta_t z_t$
2. $v_\mu(y_t) = \text{argmax}_{v \in \mathbb{Q}} \{ v^T A y_t - \mu d(v) \}$
3. Generate a random vector $\xi_t$ independently from the distribution of $\xi$.
4. $z_{t+1} = \arg\min_x \{ x^T G(y_t, \xi_t) + A^T v_\mu(y_t) \} + \frac{\mu}{2} \| x - z_t \|^2$
5. $x_{t+1} = (1 - \theta_t)x_t + \theta_t z_{t+1}$

**Output:** the solution from the $(T+1)$th iteration $x_{T+1}$.

**Remark 3** Similar to the step 3 in Algorithm 1, Algorithm 2 also has to solve a proximal mapping in Step 4. However, since $\phi_\mu(x)$ does not contain a non-smooth term like $h(x)$ in (3), the proximal mapping in Step 4 is simply an unconstrained quadratic programming whose optimal solution has a closed form:

$$z_{t+1} = z_t - \frac{1}{\gamma_t L_\mu} (G(y_t, \xi_t) + A^T v_\mu(y_t)).$$

Since Algorithm 2 just solves an approximation of (3), we have to make the smooth parameter $\mu$ small enough in order to make the solution returned by Algorithm 2 a near-optimal one for (3). However, according to (19) and Theorem 1, decreasing smooth parameter $\mu$ will increase the Lipschitz constant $L_\mu$ and more iterations will be needed in Algorithm 2 in order to minimize $\phi_\mu(x)$. Fortunately, by Theorem 1, the Lipschitz constant $L_\mu$ only appears in the $O(L_\mu / T^2)$ component of the convergence rate, which is dominated by the $O(1/T^{1/2})$ component. This means that, as long as $\mu = O(1/T^\gamma)$ with $\gamma \leq \frac{3}{2}$, which implies $L_\mu = O(T^\gamma)$ with $\gamma \leq \frac{3}{2}$, the convergence rate of Algorithm 2 is still $O(1/T^{1/2})$. Based on this observation, we prove the following convergence result for Algorithm 2 when $\mu = O(1/T)$. Similar results can be found in [12,19].

**Theorem 2** If we set $\mu = \|A\|/(T + 2)$ in Algorithm 2, then after $T$ iterations, we will have:

$$\mathbb{E}\phi(x_{T+1}) - \phi(x^*) \leq \frac{2D^2 + \sigma^2}{(T + 2)^{1/2}} + \frac{4D^2 + 2\sigma^2}{(T + 2)^2} + \frac{\|A\|}{(T + 2)} \left( M + \frac{4D^2 + 2\sigma^2}{c} \right).$$

**Proof** Because $\phi(x_{T+1}) - \phi_\mu(x_{T+1}) \leq \mu M$ and $\phi_\mu(x^*) - \phi(x^*) \leq 0$, we have

$$\mathbb{E}\phi(x_{T+1}) - \phi(x^*) = \mathbb{E}\phi(x_{T+1}) - \phi_\mu(x_{T+1}) + \mathbb{E}\phi_\mu(x_{T+1}) - \phi_\mu(x^*) + \phi_\mu(x^*) - \phi(x^*)$$

$$\leq \mu M + \mathbb{E}\phi_\mu(x_{T+1}) - \phi_\mu(x^*)$$

$$\leq \mu M + \frac{2D^2 + \sigma^2}{(T + 2)^{1/2}} + \left( L + \frac{1}{c\mu} \|A\|^2 \right) \frac{4D^2 + 2\sigma^2}{(T + 2)^2}. \quad \text{(20)}$$

where the last inequality is by Theorem 1 and (19). Setting $\mu = \|A\|/(T + 2)$, we have

$$\mathbb{E}\phi(x_{T+1}) - \phi(x^*) \leq \frac{2D^2 + \sigma^2}{(T + 2)^{1/2}} + \frac{4D^2 + 2\sigma^2}{(T + 2)^2} + \frac{\|A\|}{(T + 2)} \left( M + \frac{4D^2 + 2\sigma^2}{c} \right).$$

□
Remark 4  This theorem shows a difference between exact gradient descent and stochastic descent algorithm when smoothing technique is applied. The gradient descent algorithm proposed in [19] obtains a convergence rate of $O(1/T^2)$ but it has to be reduced to $O(1/T)$ after applying the smoothing technique. However, for Algorithm 2, smoothing technique only slows down a non-dominating component in the convergence rate such that Algorithm 2 still obtains a convergence rate of $O(1/T^{1/2})$ which is the same as Algorithm 1. In other words, the price paid for incorporating a smoothing technique is negligible.

Suppose the smooth component $f(x)$ in the objective function is not just convex but also strongly convex, the stochastic gradient algorithms developed in [6,8] can achieve a convergence rate of $O(1/T)$. Similar to the only convex cases, this convergence rate consists of two components, one term of $O(L/T^2)$ which is not dominating but contains the Lipschitz constant $L$ and one term of $O(1/T)$ which is the bottleneck but independent of $L$. Hence, by the same reasons as above, if we incorporate the smooth technique into the algorithms in [6,8] for strongly convex objective functions just as we did in Algorithm 2, we can obtain similar smoothing stochastic gradient algorithms with a convergence rate $O(1/T)$.

5. Numerical results

In this section, we apply our algorithms to four different types of regularized regression problems which belong to the class of problems formulated by (3). We compare our numerical results with the AC-SA algorithm proposed by Ghadimi and Lan in [6]. We used a Matlab implementation and ran the experiments on a computer with an Intel(R) Core(TM)2 Duo CPU T8300 2.40 GHz processor and 2.00 GB RAM. To make the notation be consistent with machine learning and statistics literature, we use $x_i$ to denote each $d$-dimensional input data and $y_i$ the response, for $1 \leq i \leq N$, where $N$ is the total number of data points. The regression coefficients are denoted by $\beta$, which are the parameters that we need to optimize.

5.1 Regularized linear regression with discrete probability distribution

Suppose there are $N$ data points $\{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The task of linear regression is to find the parameters $\beta \in \mathbb{R}^d$ to fit the linear model $y = \beta^T x + \epsilon$ by minimizing the average square loss function

$$f_1(\beta) = \frac{1}{2} \sum_{i=1}^N \frac{\|x_i^T \beta - y_i\|^2}{N} = \frac{1}{2N} \|X\beta - y\|^2,$$

where $X = [x_1, \ldots, x_N]^T$ and $y = [y_1, \ldots, y_N]^T$. Here, we assume each instance $(x_i, y_i)$ occurs with equal chance, i.e. with a probability $1/N$ so that $f_1(\beta)$ is essentially the $\frac{1}{2}$ multiple of the expectation of the square loss $(x^T \beta - y)^2$.

The gradient of $f_1(\beta)$ is

$$\nabla f_1(\beta) = \frac{1}{N} X^T (X\beta - y).$$

It is easy to prove that $\nabla f_1(\beta)$ is Lipschitz continuous with a Lipschitz constant $L = \lambda_{\text{max}}(X^TX)$. Here $\lambda_{\text{max}}(X^TX)$ denotes the largest eigenvalue of matrix $X^TX$.

Since we are testing stochastic gradient descent algorithms, we have to generate the stochastic gradient for $f_1(\beta)$ in each iteration. We first randomly sample a subset $\{(x_i, y_i)\}_{i \in S}$ with $S \subset$
\{1, 2, \ldots, N\} from the whole data set and the stochastic gradient \(G_1(\beta, S)\) corresponding to this subset is

\[
G_1(\beta, S) = \frac{1}{|S|} \mathbf{X}_S^T(\mathbf{X}_S \beta - \mathbf{y}_S),
\]

where \(\mathbf{X}_S\) and \(\mathbf{y}_S\) are sub-matrices of \(\mathbf{X}\) and \(\mathbf{Y}\) whose rows are indexed by the elements of \(S\).

When the data points belong to a high dimensional space, we are interested in selecting a small number of input features of the data which contribute most to influence the output. Hence, we want to minimize \(f_1(\beta)\) with a regularization term \(\Omega (\beta)\) which forces a highly sparse \(\beta\) with zeros in the components corresponding to the less relevant input features. Then, the regularized linear regression problem is defined as

\[
\min_{\beta} f_1(\beta) + \lambda \Omega (\beta),
\]

where \(\lambda\) is the parameter that controls the regularization level.

In our numerical experiments, we consider two different choices of \(\Omega (\beta)\). One choice is simply the \(\ell_1\)-norm of \(\beta\), i.e.

\[
\Omega_1(\beta) = \|\beta\|_1.
\]

A linear regression problem regularized by \(\Omega_1(\beta)\) is also known as the Lasso problem [23].

We apply Algorithm 1 (SG) and Algorithm 2 (SSG) proposed in this paper, the AC-SA algorithm in [6], and the Algorithm 1 (a batch deterministic gradient method) in [26] to problem (23) with \(\Omega (\beta) = \Omega_1(\beta)\). In all experiments the parameter \(\mu\) in Algorithm 2 (SSG) is set to \(\mu = \|A\|/(T + 2)\) according to Theorem 2. In this case, the proximal mappings in SG, AC-SA and the batch method have a closed form solution (see [13]). In order to apply SSG, we observe that the non-smooth term in the objective function can be represented as \(\lambda \Omega_1(\beta) = \max_{\|\alpha\|_\infty \leq 1} \alpha^T A \beta\) where \(A = \lambda I\) and \(I\) denotes the identity matrix. We choose \(d(\alpha) = \frac{1}{2} \|\alpha\|_2^2\) as the strongly convex function in SSG. With our choice of \(d(\alpha)\), the step 2 of Algorithm 2 for a given \(\beta\) can be written as the following optimization problem:

\[
v_\mu(\beta) = \arg \max_{\|\alpha\|_\infty \leq 1} \left\{ \lambda \alpha^T \beta - \frac{\mu}{2} \|\alpha\|_2^2 \right\}.
\]

It admits a closed-form solution as follows:

\[
v_\mu(\beta) = S_\infty \left( \frac{\lambda \beta}{\mu} \right),
\]

where \(S_\infty\) denotes the projection operator on to the \(\ell_\infty\)-ball:

\[
S_\infty(a) = \begin{cases} 
  a & \text{if } -1 \leq a \leq 1, \\
  1 & \text{if } a > 1, \\
  -1 & \text{if } a < -1.
\end{cases}
\]

For any vector \(\alpha\), \(S_\infty(\alpha)\) is defined as applying the operator \(S_\infty\) on each element of \(\alpha\).

We randomly generate a data set \(\{(\mathbf{x}_i, y_i)\}_{i=1}^N\) as follows. First of all, we choose the real parameter \(\tilde{\beta} \in \mathbb{R}^d\) to be \(\begin{bmatrix} 1, 1, \ldots, 1, 0, 0, \ldots, 0 \end{bmatrix}^T\) with first \(d/2\) components equal to 1 and last \(d/2\) components equal to 0. And then, we generate each data point \(\mathbf{x}_i \in \mathbb{R}^d\) by generating each of its component \(x_{ij}\) from a standard normal distribution \(N(0, 1)\) independently and we generate \(y_i\) by setting \(y_i = \mathbf{x}_i^T \tilde{\beta} + \epsilon_i/10\) with \(\epsilon_i\) generated from a standard normal distribution \(N(0, 1)\).
We generate a set of data as above with $N = 1000$ and $d = 20$ and we set the parameters $\lambda = 0.1$ and the total number of iterations $T = 50,000$. In each iteration, we randomly sample 10 data points, i.e. $|S| = 10$, to generate the stochastic gradient $G(\beta, S)$ by (22). The numerical performances of these three algorithms are shown in the left figure in Figure 1(a). The horizontal axis represents the CPU running time in the left figure and represents the number of iteration in the right figure. Here, the number of iterations indicates the number of loops of the algorithm (either stochastic or the batch algorithm). The vertical axis represents the value of objective function (23) with $\Omega(\beta) = \Omega_1(\beta)$.

Similarly, we apply these four algorithms to problem (23) with $\Omega(\beta) = \Omega_1(\beta)$ on a larger data set with $N = 100,000$ and $d = 200$. We still set $\lambda = 0.1$ and $T = 50,000$ but we increase the sample size $|S|$ to 100. The numerical result is shown in Figure 1(b).

The other choice for $\Omega(\beta)$ is the overlapped group sparsity inducing norm introduced by Jenatton et al. [10]. Suppose the set of groups of inputs $\mathcal{G} = \{g_1, \ldots, g_{|G|}\}$ is a subset of the power set of $\{1, 2, \ldots, d\}$, the overlapped group sparsity inducing norm $\Omega_2(\beta)$ is defined as

$$
\Omega_2(\beta) = \sum_{g \in \mathcal{G}} w_g \|\beta_g\|,
$$

where $\beta_g \in \mathbb{R}^{|g|}$ is a sub-vector of $\beta$ which only contains the components of $\beta$ indexed by the elements of $g$ and $w_g$ is a positive constant for each $g \in \mathcal{G}$. 

Figure 1. Linear regression with $\ell_1$-norm regularization. (a) $N = 1000$, $d = 20$ and (b) $N = 100,000$, $d = 200.$
To be specific, in our numerical experiments, we set \( d = 2^n \) for a positive integer \( n \) and \( w_\theta = \sqrt{\|g\|} \) and we define the set of groups of inputs \( G \) as follows

\[
G = \left\{ \begin{array}{c}
\{g_{0,1}, g_{0,2}, \ldots, g_{0,2^n}\} \\
\{g_{1,1}, g_{1,2}, \ldots, g_{1,2^n-1}\} \\
\vdots \\
\{g_{1,1}, \ldots, g_{1,2^n-2}\} \\
\{g_{n,1}\}
\end{array} \right\},
\]

where

\[
g_{i,j} = \{(j - 1)2^i + 1, (j - 1)2^i + 2, \ldots, j2^i\} \quad \text{for} \quad i = 1, 2, \ldots, n \quad \text{and} \quad j = 1, 2, \ldots, 2^n - i.
\]

This particular type of overlapped group sparsity inducing norm is also called \textit{hierarchical norm} \cite{10}.

We apply Algorithms SG, SSG, AC-SA and the batch method to the problem \((23)\) with \( \Omega(\beta) = \Omega_2(\beta) \). In this case, the proximal mappings in SG, AC-SA and the batch algorithms no longer have a closed form solutions. Jenatton et al. \cite{21} propose a coordinate descent method which can solve the proximal mappings when we apply SG, AC-SA and the batch method to the hierarchical norm regularized regression problem.

In order to apply SSG, we need to reformulate the non-smooth term \( \lambda \Omega_2(\beta) \) with formulation \((2)\). Since the dual norm of Euclidean norm is Euclidean norm itself, \( \|\beta\| = \max_{\|\alpha\| \leq 1} \|\alpha^T\beta\| \), where \( \alpha_\theta \in \mathbb{R}^{|g|} \) is the vector of auxiliary variables associated to \( \beta_\theta \). Let \( \alpha = [\alpha_{g_1}, \ldots, \alpha_{g_{|G|}}]^T \) be the vector of length \( \sum_{g \in G} |g| \) and denote the domain of \( \alpha \) by \( Q = \{\alpha \ | \|\alpha_\theta\| \leq 1, \forall \theta \in G\} \). Note that \( Q \) is the Cartesian product of unit balls in Euclidean space which is a closed and convex set. We can rewrite \( \lambda \Omega_2(\beta) \) as:

\[
\lambda \Omega_2(\beta) = \lambda \sum_{g \in G} w_\theta \max_{\|\alpha_\theta\| \leq 1} \|\alpha_\theta^T\beta_\theta\| = \max_{\alpha \in Q} \sum_{g \in G} \lambda w_\theta \|\alpha_\theta^T\beta_\theta\| = \max_{\alpha \in Q} \|\alpha^T A \beta\|,
\]

where \( A \in \mathbb{R}^{\sum_{g \in G} |g| \times d} \) is a matrix such that \( A \beta = [\lambda w_{g_1} \beta_{g_1}^T, \ldots, \lambda w_{g_{|G|}} \beta_{g_{|G|}}^T]^T \). The rows of \( A \) are indexed by all pairs of \((i, g)\) such that \( i \in \{1, \ldots, d\}, i \in g \) and its columns are indexed by \( j \in \{1, \ldots, d\} \) and \( A \) is defined as:

\[
A_{(i, \theta), j} = \begin{cases} 
\lambda w_\theta & \text{if } i = j, \\
0 & \text{otherwise}.
\end{cases}
\]

By choosing \( d(\alpha) = \frac{1}{2} \|\alpha\|^2 \), Step 2 of Algorithm 2 for a given \( \beta \) admits a closed-form solution as shown in Proposition 1 in \cite{4}. For the purpose of completeness, we present the optimal solution \( v_\mu(\beta) \). It is the concatenation of \( [v_\mu(\beta)]_g \) for each group \( g \), where

\[
[v_\mu(\beta)]_g = S_2 \left( \frac{\lambda w_\theta \beta_\theta}{\mu} \right).
\]

Here \( S_2 \) denotes the projection operator on to the \( \ell_2 \)-ball:

\[
S_2(\alpha) = \begin{cases} 
\alpha & \text{if } \|\alpha\| \leq 1, \\
\frac{\alpha}{\|\alpha\|} & \text{if } \|\alpha\| > 1.
\end{cases}
\]

Different from the other three methods, the proximal mapping in SSG always has a closed form solution.
We generate a data set \( \{(x_i, y_i)\}_{i=1}^{N} \) in the same way as before with \( N = 1000 \) and \( n = 5 \) (\( d = 2^n = 32 \)) and set the parameters \( \lambda = 0.1 \), the total number of iterations \( T = 10,000 \) and the sample size \( |S| = 10 \). The numerical results by these algorithms on this data set are shown in Figure 2(a).

Also, we generate a larger data set with \( N = 100,000 \) and \( n = 9 \) (\( d = 2^n = 512 \)) and run the algorithms on it with \( \lambda = 0.1 \), \( T = 10,000 \) and \( |S| = 100 \). The numerical results are posted in Figure 2(b).

From Figures 1 and 2, we can imply that even though the SG, SSG and AC-SA have the same \( O(1/\sqrt{T}) \) theoretical convergence rate, their performances are different in practical applications. Our SG and SSG methods have better performances than AC-SA method in these particular numerical settings. One of the reasons for this result is that AC-SA algorithm chooses \( 1/\gamma_t L = O(t/T^{3/2}) \) (see Remark 1 or Proposition 4 in [6]) as its step length, which is very small when \( T \) is large, in order to mitigate the impact from the inaccuracy of the stochastic gradient. This is needed in the theoretical proof of the convergence rate of AC-SA. However, when the stochastic gradient is a good approximation for the exact gradient, e.g. \( \sigma^2 \) is very small, such a small step length in AC-SA is too conservative. Instead, SG and SSG adopt a relatively larger step length such that they can reduce the objective functions more efficiently.

The influence of the smoothing technique by Nesterov [19] is also reflected by these numerical results. When \( \Omega_1(\beta) \) is chosen as the regularization term, the proximal mapping in SG has a closed form solution so that applying the smoothing technique does not bring additional saving in time for each iteration. Hence, in Figure 1, the blue curve (SG) and the green curve (SSG) almost
overlap. This is in compliance with the fact that SG and SSG have a same $O(1/\sqrt{T})$ complexity shown by Theorems 1 and 2.

However, in Figure 2, we can see that SSG is much more efficient than SG in terms of CPU time. This is because SG has to use a coordinate descent method to solve the proximal mapping due to the lack of closed form solution when $\Omega_2(\beta)$ is the regularization term. Even though the coordinate descent method is shown to converge after finite steps, it is still not necessarily faster than solving it by a closed form which is available in SSG.

In both Figures 1 and 2, the batch method always outperforms all stochastic gradient methods in terms of the number of iterations. This is because the batch method utilizes the exact gradient and achieves a $O(1/T^2)$ convergence rate while the three stochastic gradient methods only have a convergence rate of $O(1/\sqrt{T})$ due to the noisy gradient information. If evaluated in terms of CPU time, the batch method only outperforms when the problem size is small ($N = 1000$ and $d = 20$), since, in this case, the time it spends on each iteration is comparable to that of a stochastic gradient method. However, for a large problem ($N = 100,000$ and $d = 200$), SG and SSG become more efficient due to the large time cost per iteration in the batch method.

5.2 Regularized linear regression with continuous probability distribution

We apply our algorithms again to regularized linear regression problems. Here, we assume that there are infinitely many data points $(x, y) \in \mathbb{R}^{d+1}$ which follow a continuous distribution $p(x, y)$. The task is still to find the parameters $\beta \in \mathbb{R}^d$ to fit the linear model $y = \beta^T x + \epsilon$ by minimizing the average square loss function

$$f_1^c(\beta) = \frac{1}{2} \mathbb{E}(x^T \beta - y)^2 = \frac{1}{2} \int (x^T \beta - y)^2 p(x, y) \, dx \, dy,$$

which can be viewed as $f_1(\beta)$ in (21) when $N = \infty$.

In our numerical experiment, we make $x$ follow the standard normal distribution in $\mathbb{R}^d$, i.e. $p(x) = N(0, I)$. The real parameters $\hat{\beta}$ are chosen to be $[1, 1, \ldots, 1, 0, 0, \ldots, 0]^T$ with first $d/2$ components equal to 1 and last $d/2$ components equal to 0 and the error term $\epsilon$ in the linear model is assumed to have a standard normal distribution $N(0, 1)$ so that the variable $y$ follows a normal distribution $p(y \mid x) = N(x^T \hat{\beta}, 1)$ once $x$ is fixed. By these settings, the distribution $p(x, y)$ in our numerical experiments is

$$p(x, y) = p(x)p(y \mid x) = \frac{1}{(2\pi)^{d/2}} e^{-(1/2)x^T x} \frac{1}{(2\pi)^{1/2}} e^{-(1/2)(y-\hat{\beta}^T x)^2} \frac{1}{(2\pi)^{(d+1)/2}} e^{-(x^T x + (y-\hat{\beta}^T x)^2)/2}.$$

It is easy to show that, in this case, the loss function $f_1^c(\beta)$ becomes

$$f_1^c(\beta) = \frac{1}{2} \left( \beta^T \beta - 2 \beta^T \hat{\beta} + \frac{d}{2} + 1 \right),$$

whose gradient is simply $\nabla f_1^c(\beta) = \beta - \hat{\beta}$ with a Lipschitz constant $L = 1$.

Similar to the discrete cases, we apply our algorithms to the following regularized linear regression problem

$$\min_{\beta} f_1^c(\beta) + \lambda \Omega(\beta),$$

where $\Omega(\beta)$ is the regularization term.
For the batch algorithm, we note that it does not mean that we sample $N = +\infty$ data points for each iteration. In fact, the batch algorithm directly minimizes (34) with the given loss function $f^{ij}_c(\beta)$ in (33). Such an explicit loss is a function of only $\beta$ and is independent of the data. For stochastic optimization algorithms, the explicit form of $f^{ij}_c(\beta)$ is unavailable and can only be approximated by sampling the data. In order to generate a stochastic approximation for $\nabla f^{ij}_c(\beta)$, in each iteration, we sample a set of points $S = \{(x_i, y_i)\}_{i=1,\ldots,|S|}$ by generating $x_i$ from $N(0, I)$ and $\epsilon_i$ from $N(0, 1)$ and setting $y_i = x_i^T \hat{\beta} + \epsilon_i$ for $i = 1, \ldots, |S|$. Then we can compute a stochastic gradient $G(\beta, S)$ by (22).

We compare Algorithms AC-SA, SG, SSG and the batch algorithms and present their performance in Figures 3 and 4. In Figure 3, we set $d = 1000$, $|S| = 10$, $\lambda = 0.1$ and $T = 10,000$ and the vertical axis represents the value of objective function (34) with $\Omega(\beta) = \Omega_1(\beta)$. In Figure 4, we set $n = 8$ ($d = 2^n = 256$), $|S| = 100$, $\lambda = 0.1$ and $T = 1000$ and the vertical axis represents the value of objective function (34) with $\Omega(\beta) = \Omega_2(\beta)$.

The plots on the left in Figures 3 and 4 reflect similar phenomena as the plots on the left in Figures 1 and 2. SG and SSG converge faster than AC-SA in CPU time and, when the regularization term is complicated, SSG significantly outperforms the other two algorithms in CPU time. However, AC-SA can outperform SG and SSG (see the right figure in Figure 4) in terms of the number of iterations. The reason is that, in this setting, the total number of iteration $T = 1000$ here is not as large as those used in Figure 1 ($T = 50,000$) and Figure 2 ($T = 10,000$). As a result, the step length of AC-SA is larger and less conservative so that AC-SA is not necessarily worse than
SG and SSG in the efficiency of each iteration. The batch method always have the best performance in the number of iterations due to its better theoretical convergence rate. However, it does not significantly outperform SSG in CPU time with the hierarchical norm regularization. This is again because the smoothing technique significantly reduces the time cost per iteration in SSG.

5.3 Convergence rate with different sample sizes

In order to understand how the convergence of our SSG method depends on the size of the sample which the stochastic gradient is built on, we apply SSG method again on the regularized linear regression problem with a continuous probability distribution, i.e. (34), with \( \Omega(\beta) = \Omega_2(\beta) \) and \( n = 8 \) (\( d = 2^n = 256 \)). We implement SSG with three different sizes of the data sample, \(|S| = 1, 10, 100\), and show the numerical comparison in Figure 5.

According to the plot on the right in Figure 5, as the sample size increases, SSG method achieves a smaller objective value with the same number of iterations. This is fully supported by Theorem 2 since a larger \( S \) implies a smaller noise level \( \sigma \) in the stochastic gradient, and thus, a smaller right hand side in the conclusion of Theorem 2. In this figure, it is also shown that, SSG method converges poorly when \( |S| = 1 \) because the gradient information contains a large noise with such a small sample.

An interesting phenomenon is shown by the plot on the left in Figure 5. The SSG method performs the best in terms of CPU time when the sample size is medium (\( |S| = 10 \)). The explanation is that, when the sample size is too small (\( |S| = 1 \)), each iteration is very fast but not efficient to push the solution towards the optimality due to the large gradient noise. When the sample size is too large (\( |S| = 100 \)), SSG is slowed down by its per iteration cost although its iteration can reduce the objective value more efficiently.

5.4 Regularized logistic regression

Suppose there are \( N \) data points \( \{(x_i, y_i)\}_{i=1}^N \), where each input \( x_i \in \mathbb{R}^d \) is normalized to have the Euclidean norm 1 and \( y_i \in \{0, 1\} \) is the class label of \( x_i \) which indicates that \( x_i \) belongs to class 0 or class 1. We assume that the posterior probability of the class label of a particular input \( x \) is given by

\[
\Pr(y = 1 \mid x) = \frac{e^{\beta^T x}}{1 + e^{\beta^T x}},
\]

(35)
The batch method still performs the best in this setting because the problem sizes are not large enough.

AC-SA have to rely on another algorithm as a subroutine to solve their proximal mappings. Similar to the regularized linear regression problem, we minimize $f_2(\beta)$ together with a regularization term $\Omega(\beta)$ in order to obtain a sparse solution $\beta$. Hence, the regularized logistic regression can be formulated as

$$\min_{\beta} f_2(\beta) + \lambda \Omega(\beta),$$

where $\Omega(\beta)$ can also be chosen to be $\Omega_1(\beta)$ or the hierarchical norm $\Omega_2(\beta)$ defined by (26)–(28).

The gradient of $f_2(\beta)$ is the following

$$\nabla f_2(\beta) = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} - y_i \right) x_i.$$

Because each data point satisfies $\|x_i\| = 1$, it can be shown that $\nabla f_2(\beta)$ is Lipschitz continuous with a Lipschitz constant $L = 1$. Similar to the regularized linear regression problems, we randomly sample a subset $\{(x_i, y_i)\}_{i \in S}$ with $S \subseteq \{1, 2, \ldots, N\}$ from the whole data set and generate the stochastic gradient $G_2(\beta, S)$ for $f_2(\beta)$ as follows

$$G_2(\beta, S) = \frac{1}{|S|} \sum_{i \in S} \left( \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} - y_i \right) x_i.$$

Now we apply Algorithms SG, SSG, AC-SA and the batch method to problem (37) with $\Omega(\beta) = \Omega_1(\beta)$. We create a set of artificial data $\{(x_i, y_i)\}_{i=1}^{N}$ with $N = 1000$ and $d = 20$ as follows. At first, we choose the real parameter $\hat{\beta}$ to be an all-ones vector in $\mathbb{R}^d$. After that, for each $i = 1, \ldots, N$, we create a $\hat{x}_i$ by generating each of its component $\hat{x}_{ij}$ from a standard normal distribution $N(0, 1)$ independently and we get $x_i$ by normalizing $\hat{x}_i$, i.e. $x_i = \hat{x}_i / \|\hat{x}_i\|$. The corresponding $y_i$ is set to be 1 or 0 randomly with the probabilities defined by (35). We set the sample size $|S| = 10$, $\lambda = 0.01$ and the number of iterations $T = 50,000$ in all of the three algorithms. The numerical performances are presented in Figure 6.

For the problem (37) with $\Omega(\beta) = \Omega_2(\beta)$, we generate the data in the same way as above but with $N = 1000$ and $n = 5$ ($d = 32$). Still, $|S|$, $\lambda$ and $T$ are set to be 10, 0.01 and 50,000 respectively. We put the curves in Figure 7 to show how the objective values decrease in these algorithms.

The properties of our algorithms shown by Figures 6 and 7 are very similar to what are shown in Figures 1–4. In the $\ell_1$-norm regularized logistic regression problems, SG and SSG are more efficient than AC-SA due to the more aggressive choices of the step lengths. In the cases of hierarchical norm regularized logistic regression, SSG is more efficient in terms of CPU time than the other two just because SSG has a closed form solution for its proximal mapping but SG and AC-SA have to rely on another algorithm as a subroutine to solve their proximal mappings. The batch method still performs the best in this setting because the problem sizes are not large enough.
6. Conclusion

In this paper, we consider an optimization problem whose objective function is a composition of a smooth convex function and a non-smooth convex function. We first developed a stochastic gradient descent algorithm for solving this problem. We also proposed another stochastic gradient descent algorithm by smoothing the non-smooth term in the objective function. The convergence rates of these two algorithm are proved. The results of our numerical experiments demonstrate efficiency and scalability of our algorithms.

Note

1. In this paper, the notation $\| \cdot \|$ without any subscript represents the Euclidean norm of a vector.

References

[1] A. Auslender and M. Teboulle, Interior gradient and proximal methods for convex and conic optimization, SIAM J. Optim. 16 (2006), pp. 697–725.
[2] A. Beck and M. Teboulle, A fast iterative shrinkage-threshold algorithm for linear inverse problems, SIAM J. Optim. 2 (2009), pp. 183–202.
[3] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, NY, 2003.
[4] X. Chen, Q. Lin, S. Kim, J.G. Carbonell, and E.P. Xing, *Smoothing proximal gradient method for general structured sparse learning*, Ann. Appl. Stat. 6 (2012), pp. 719–752.

[5] J. Friedman, T. Hastie, and R. Tibshirani, *Regularization paths for generalized linear models via coordinate descent*, J. Stat. Softw. 33(1) (2010), pp. 1–22.

[6] S. Ghadimi and G. Lan, *Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization, I: A generic algorithmic framework*, SIAM J. Optim. 22 (2012), pp. 1469–1492.

[7] J.-B. Hiriart-Urruty and C. Lemarechal, *Fundamentals of Convex Analysis*, Springer, Heidelberg, 2001.

[8] C. Hu, J.T. Kwok, and W. Pan, *Accelerated gradient methods for stochastic optimization and online learning*, Adv. Neural Inf. Process. Syst. 22 (2009), pp. 781–789.

[9] L. Jacob, G. Obozinski, and J.-P. Vert, *Group Lasso with overlap and graph Lasso*, International Conference on Machine Learning, Montreal, QC, Canada, 2009.

[10] R. Jenatton, J.-Y. Audibert, and F. Bach, *Structured variable selection with sparsity-inducing norms*, J. Mach. Learn. Res. 12 (2011), pp. 2777–2824.

[11] G. Lan, *An optimal method for stochastic composite optimization*, Math. Program. 133(1) (2012), pp. 365–397.

[12] G. Lan, Z. Lu, and R.D.C. Monteiro, *Primal-dual first-order methods with o(1/ε) iteration-complexity for cone programming*, Math. Program. 126(1) (2009), pp. 1–29.

[13] J. Liu, S. Ji, and J. Ye, *Multi-task feature learning via efficient ℓ₁/ℓ₂ norm minimization*, Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, 2009.

[14] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, *Applications of second-order cone programming*, Linear Algebra Appl. 284 (1998), pp. 193–228.

[15] L. Meier, S. van de Geer, and P. Bühlmann, *The group Lasso for logistic regression*, J. R. Stat. Soc. Ser. B 70 (2008), pp. 53–71.

[16] A. Nemirovski and D. Yudin, *Problem Complexity and Method Efficiency in Optimization*, John Wiley, New York, 1983.

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

[18] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Kluwer Academic Pub, Norwell, MA, 2003.

[19] Y. Nesterov, *Smooth minimization of non-smooth functions*, Math. Program. 103(1) (2005), pp. 127–152.

[20] Y. Nesterov, *Gradient methods for minimizing composite objective function*, Math. Program. 140(1) (2013), pp. 125–161.

[21] G. Obozinski, F. Bach, R. Jenatton, and J. Mairal, *Proximal methods for sparse hierarchical dictionary learning*, International Conference on Machine Learning, Haifa, Israel, 2010.

[22] T.K. Pong, P. Tseng, S. Ji, and J. Ye, *Trace norm regularization: Reformulations, algorithms, and multi-task learning*, SIAM J. Optim. 20(6) (2010), pp. 3465–3489.

[23] R. Tibshirani, *Regression shrinkage and selection via the Lasso*, J. R. Stat. Soc. Ser. B 58 (1996), pp. 267–288.

[24] R. Tibshirani and M. Saunders, *Sparsity and smoothness via the fused Lasso*, J. R. Stat. Soc. Ser. B 67(1) (2005), pp. 91–108.

[25] K.-C. Toh and S. Yun, *An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems*, Pac. J. Optim. 6(3) (2010), pp. 615–640.

[26] P. Tseng, *On accelerated proximal gradient methods for convex-concave optimization*, Tech. Rep. Department of Mathematics at the University of Washington, Seattle, Washington, 2008. Available at http://pages.cs.wisc.edu/~brecht/cs726docs/Tseng.APG.pdf

[27] P. Tseng and S. Yun, *A coordinate gradient descent method for nonsmooth separable minimization*, Math. Program. 117 (2009), pp. 387–423.

[28] M. Yuan and Y. Lin, *Model selection and estimation in regression with grouped variables*, J. R. Stat. Soc. Ser. B 68 (2006), pp. 49–67.