INVITED PAPER

Variance-Reduced Methods for Machine Learning

This article discusses stochastic variance-reduced optimization methods for problems where multiple passes through batch training data sets are allowed.

By Robert M. Gower, Mark Schmidt, Francis Bach, and Peter Richtárik

ABSTRACT | Stochastic optimization lies at the heart of machine learning, and its cornerstone is stochastic gradient descent (SGD), a method introduced over 60 years ago. The last eight years have seen an exciting new development: variance reduction for stochastic optimization methods. These variance-reduced (VR) methods excel in settings where more than one pass through the training data is allowed, achieving a faster convergence than SGD in theory and practice. These speedups underline the surge of interest in VR methods and the fast-growing body of work on this topic. This review covers the key principles and main developments behind VR methods for optimization with finite data sets and is aimed at nonexpert readers. We focus mainly on the convex setting and leave pointers to readers interested in extensions for minimizing nonconvex functions.

KEYWORDS | Machine learning; optimization; variance reduction.

I. INTRODUCTION

One of the fundamental problems studied in the field of machine learning is how to fit models to large data sets. For example, consider the classic linear least squares model

$$x_\star \in \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{n}\sum_{i=1}^{n}(a_i^\top x - b_i)^2 \right\}. \tag{1}$$

Here, the model has d parameters given by the vector $x \in \mathbb{R}^d$, and we are given n data points $\{a_i, b_i\}$ consisting of feature vectors $a_i \in \mathbb{R}^d$ and target values (labels) $b_i \in \mathbb{R}$. Fitting the model consists of tuning these d parameters so that the model's output $a_i^\top x$ is "close" (on average) to the targets b_i. More generally, we might use some loss function $f_i(x)$ to measure how close our model is to the ith data point

$$x_\star \in \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \left\{ f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) \right\}. \tag{2}$$

If $f_i(x)$ is large, we say that our model's output is far from the data, and if $f_i(x) = 0$, we say that our model perfectly fits the ith data point. The function $f(x)$ represents the average loss of our model over the full data set. A problem of the form (2) characterizes the training of not only linear least squares, but also many models studied in machine learning. For example, the logistic regression model solves

$$x_\star \in \underset{x\in\mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-b_i a_i^\top x)) + \frac{\lambda}{2}\|x\|^2 \right\} \tag{3}$$

where we are now considering a binary classification task with $b_i \in \{-1, +1\}$ (and the predictions are made using the sign of $a_i^\top x$). Here, we have also used $(\lambda/2)\|x\|^2 := (\lambda/2)\sum_{i=1}^{d} x_i^2$ as a regularizer. This and other regularizers are commonly added to avoid overfitting to the given data, and in this case, we replace each $f_i(x)$ by $f_i(x)+(\lambda/2)\|x\|^2$. The training procedure in most supervised machine learning models can be written in the form (2), including L1-regularized least squares, support vector machines (SVM), principal component analysis, conditional random fields, and deep neural networks.

A key challenge in modern instances of problem (2) is that the number of data points n can be extremely large. We regularly collect the data sets going beyond terabytes, from sources such as the Internet, satellites, remote sensors, financial markets, and scientific experiments. One of the most common ways to cope with such large data sets

Manuscript received January 8, 2020; revised May 8, 2020; accepted June 8, 2020. Date of current version October 27, 2020. This work was supported in part by the Canada CIFAR AI Chair Program and in part by the KAUST Research Baseline Scheme. *(Corresponding author: Robert M. Gower.)*

Robert M. Gower is with LTCI, Télécom Paris, Institut Polytechnique de Paris, 75634 Paris, France (e-mail: gower.robert@gmail.com).

Mark Schmidt is with CCAI Affiliate Chair (Amii), The University of British Columbia, Vancouver, BC V6T 1Z4, Canada.

Francis Bach is with Inria, PSL Research University, 75006 Paris, France.

Peter Richtárik is with the Department of Computer Science, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia.

Digital Object Identifier 10.1109/JPROC.2020.3028013

Stochastic variance-reduced methods are as cheap to update as SGD, and also have a fast exponential convergence like full gradient descent.

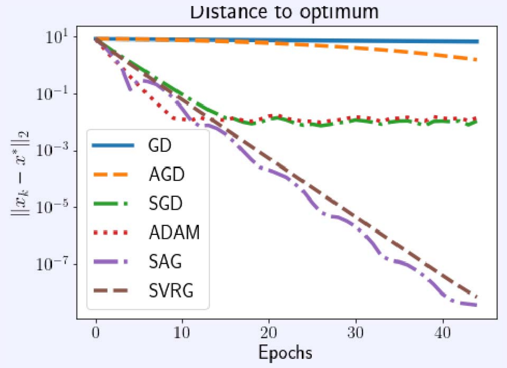

Fig. 1. ***Comparison of the GD, AGD (accelerated GD [50]), SGD, and ADAM [30] methods to the VR methods SAG and SVRG on a logistic regression problem based on the mushrooms data set [7], where n = 8124 and d = 112.***

is to use stochastic gradient descent (SGD) methods that use a few randomly chosen data points in each of their iterations. Furthermore, there has been a recent surge in interest in variance-reduced (VR) stochastic gradient methods that converge faster than classic stochastic gradient methods.

A. Gradient and Stochastic Gradient Descent

The classic gradient descent (GD) method applied to problem (2) takes the form

$$x_{k+1} = x_k - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_k) \tag{4}$$

where $\gamma > 0$ is a fixed stepsize.[1] At each iteration, the GD method needs to calculate a gradient $\nabla f_i(x_k)$ for every ith data point, and thus, GD takes a full pass over the n data points at each iteration. This expensive cost per iteration makes GD prohibitive when n is large.

Consider, instead, the SGD method

$$x_{k+1} = x_k - \gamma \nabla f_{i_k}(x_k) \tag{5}$$

first introduced by Robbins and Monro [64]. It avoids the heavy cost per iteration of GD by using one randomly selected $\nabla f_{i_k}(x_k)$ gradient instead of the full gradient. In Fig. 1, we see how the SGD method makes dramatically more progress than GD (and even the "accelerated" GD method) in the initial phase of optimization. Note that this figure plots the progress in terms of the number of epochs, which is the number of times that we have computed n gradients of individual training examples. The GD method does one iteration per epoch, while the SGD method does n iterations per epoch. We compare SGD and GD in terms of epochs taken since we assume that n is very large and that the main cost of both methods is computing the $\nabla f_i(x_k)$ gradients.

B. Issue With Variance

Observe that if we choose the random index $i_k \in \{1, \ldots, n\}$ uniformly, $\mathbf{P}[i_k = i] = \frac{1}{n}$ for all i, then $\nabla f_{i_k}(x_k)$ is an unbiased estimate of $\nabla f(x_k)$ since

$$\mathbf{E}\left[\nabla f_{i_k}(x_k) \mid x_k\right] = \sum_{i=1}^{n} \frac{1}{n} \nabla f_i(x_k) = \nabla f(x_k). \tag{6}$$

Thus, even though the SGD method is not guaranteed to decrease f in each iteration, on average, the method is moving in the direction of the negative full gradient, which is a direction of descent.

Unfortunately, having an unbiased estimator of the gradient is not enough to guarantee convergence of the iterates (5) of SGD. To illustrate this, in Fig. 2 (left), we have plotted the iterates of SGD with a constant stepsize applied to a logistic regression function using the `fourclass` data set from LIBSVM [7]. The concentric ellipses in Fig. 2 are the level sets of this function, that is, the points x on a single ellipse are given by $\{x : f(x) = c\}$ for a particular constant $c \in \mathbb{R}$. Different constants c give different ellipses.

The iterates of SGD do not converge to the solution (the green star) and, instead, form a point cloud around the solution. In contrast, we have plotted the iterates of a VR method stochastic average gradient (SAG) (which we present later) in Fig. 2 using the same constant stepsize.

The reason why SGD does not converge in this example is because the stochastic gradients themselves do no converge to zero, and thus, the method (5) with a constant stepsize never stops. This is in contrast with GD, where the method naturally stops since $\nabla f(x_k) \to 0$ as $x_k \to x_\star$.

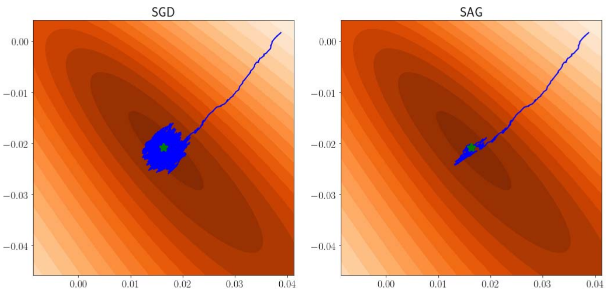

Fig. 2. ***Level set plot of 2-D logistic regression with the iterates of SGD (left) and SAG (right) with a constant stepsize. The green star is the x_* solution.***

[1] The classic way to implement GD is to determine γ as the approximate solution to $\min_{\gamma>0} f(x_k - \gamma \nabla f(x_k))$. This is called a line search since it is an optimization over a line segment [3], [45]. This line search requires multiple evaluations of the full objective function $f(x_k)$, which, in our setting, is too expensive, since this would require loading all the data points multiple times. This is why we use fixed constant stepsize instead.

C. Classic Variance Reduction Methods

There are several classic techniques for dealing with the nonconvergence due to the variance in the $\nabla f_i(x_k)$ values. For example, Robbins and Monro [64] address the issue of the variance using a sequence of decreasing stepsizes γ_k. This forces the product $\gamma_k \nabla f_{i_k}(x_k)$ to converge to zero. However, it is difficult to tune this sequence of decreasing stepsizes so that the method does not stop too early (before reaching the solution) or too late (thus wasting resources).

Another classic technique for decreasing the variance is to use the average of several $\nabla f_i(x_k)$ values in each iteration to get a better estimate of the full gradient $\nabla f(x)$. This is called minibatching and is especially useful when multiple gradients can be evaluated in parallel. This leads to an iteration of the form

$$x_{k+1} = x_k - \gamma \frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \tag{7}$$

where $B_k \subset \{1, \dots, n\}$ is a set of random indices and $|B_k|$ is the size of B_k. When B_k is sampled uniformly with replacement, the variance of this gradient estimator is inversely proportional to the "batch size" $|B_k|$, so we can decrease the variance by increasing the batch size.[2] However, the cost of this iteration is proportional to the batch size. Thus, this form of variance reduction comes at a computational cost.

Yet, another common strategy to decrease variance and improve the empirical performance of SGD is to add "momentum," an extra term based on the directions used in past steps. In particular, SGD with momentum takes the form

$$m_k = \beta m_{k-1} + \nabla f_{i_k}(x_k) \tag{8}$$

$$x_{k+1} = x_k - \gamma m_k \tag{9}$$

where the momentum parameter β is in the range $(0, 1)$. Setting $m_0 = 0$ and expanding the update of m_k in (8), we have that m_k is a weighted average of the previous gradients

$$m_k = \sum_{t=0}^{k} \beta^{k-t} \nabla f_{i_t}(x_t). \tag{10}$$

Thus, m_k is a weighted sum of the stochastic gradients. Moreover, since $\sum_{t=0}^{k} \beta^{k-t} = (1 - \beta^{k+1})/(1 - \beta)$, we have that $(1 - \beta)/(1 - \beta^{k-1}) m_k$ is a weighted average of stochastic gradients. If we compare this with the expression of the full gradient that is a plain average, $\nabla f(x_k) = (1/n) \sum_{i=1}^{n} \nabla f_i(x_k)$, we can interpret $(1 - \beta)/(1 - \beta^k) m_k$ (and m_k) as an estimate of the full gradient. This weighted sum decreases the variance, but it also brings about a key problem. Since the weighted sum (10) gives more weight to recently sampled gradients, it does not converge to the full gradient $\nabla f(x_k)$, which is a plain average. The first VR method that we will see in Section II-A contours this issue by using a plain average, as opposed to any weighted average.

D. Modern Variance Reduction Methods

As opposed to classic methods that use one or more $\nabla f_i(x_k)$ directly as an approximation of $\nabla f(x_k)$, VR methods use $\nabla f_i(x_k)$ to update an estimate $g_k \in \mathbb{R}^d$ of the gradient so that $g_k \approx \nabla f(x_k)$. With this gradient estimate, we then take approximate gradient steps of the form

$$x_{k+1} = x_k - \gamma g_k \tag{11}$$

where $\gamma > 0$ is again the stepsize. To make (11) converge with a constant stepsize, we need to ensure that the variance of our gradient estimate g_k converges to zero, that is,[3]

$$\mathbf{E}\left[\|g_k - \nabla f(x_k)\|^2\right] \xrightarrow[k \to \infty]{} 0 \tag{12}$$

where the expectation is taken with respect to all the random variables in the algorithm up to iteration k. Property (12) ensures that the VR method will stop when reaching the optimal point. We take (12) to be a defining property of VR methods and, thus, refer to it as the VR property. Note that "reduced" variance is a bit misleading since the variance converges to zero. The property (12) is responsible for the faster convergence of VR methods in theory (under suitable assumptions) and in practice as we see in Fig. 1.

E. First Example of a VR Method: SGD$_\star$

One easy fix that makes the SGD recursion in (5) converge without decreasing the stepsize is to simply shift each gradient by $\nabla f_i(x_\star)$, that is, to use the following method:

$$x_{k+1} = x_k - \gamma(\nabla f_{i_k}(x_k) - \nabla f_{i_k}(x_\star)) \tag{13}$$

called SGD$_\star$ [22]. We note that it is unrealistic that we would know each $\nabla f_i(x_\star)$, but we use SGD$_\star$ as a simple illustration of the properties of VR methods. Furthermore, many VR methods can be seen as an approximation of the SGD$_\star$ method; instead of relying on knowing each $\nabla f_i(x_\star)$, these methods use approximations that converge to $\nabla f_i(x_\star)$.

Note that SGD$_\star$ uses an unbiased estimate of the full gradient. Indeed, since $\nabla f(x_\star) = 0$,

$$\mathbf{E}\left[\nabla f_{i_k}(x_k) - \nabla f_{i_k}(x_\star)\right] = \nabla f(x_k) - \nabla f(x_\star) = \nabla f(x_k).$$

[2] If we sample without replacement, the variance decreases at a faster rate (see [42, Sec. 2.7]), and with $|B_k| = n$, the variance is zero.

[3] To be exact, (12) is not explicitly the total variance of g_k but rather the trace of the covariance matrix of g_k.

Furthermore, SGD$_\star$ naturally stops when it reaches the optimal point since; for any i,

$$(\nabla f_i(x) - \nabla f_i(x_\star))|_{x=x_\star} = 0.$$

Next, we note that SGD$_\star$ satisfies the VR property (12) as x_k approaches $x_\star$ (for continuous ∇f_i) since

$$\begin{aligned}\mathbf{E}\left[\|g_k - \nabla f(x_k)\|^2\right] &= \mathbf{E}\left[\|\nabla f_i(x_k) - \nabla f_i(x_\star) - \nabla f(x_k)\|^2\right] \\ &\leq \mathbf{E}\left[\|\nabla f_i(x_k) - \nabla f_i(x_\star)\|^2\right]\end{aligned}$$

where we used Lemma 2 with $X = \nabla f_i(x_k) - \nabla f_i(x_\star)$ and then used that $\mathbf{E}\left[\nabla f_i(x_k) - \nabla f_i(x_\star)\right] = \nabla f(x_k)$. This property implies that SGD$_\star$ has a faster convergence rate than classic SGD methods, as we detail in Appendix B.

F. Faster Convergence of VR Methods

In this section, we introduce two standard assumptions that are used to analyze VR methods and discuss the speedup over classic SGD methods that can be obtained under these assumptions. Our first assumption is Lipschitz continuity of the gradients, meaning that the gradients cannot change arbitrarily fast.

Assumption 1: The function f is differentiable and L-smooth, meaning that

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{14}$$

for all x and y and some $0 < L < \infty$. Each $f_i : \mathbb{R}^d \to \mathbb{R}$ is differentiable, L_i-smooth, and let $L_{\max} := \max\{L_1, \ldots, L_n\}$.

While this is typically viewed as a weak assumption, in Section IV, we comment on VR methods that apply to nonsmooth problems. This L-smoothness assumption has an intuitive interpretation for univariate functions that are twice differentiable: it is equivalent to assuming that the second derivative is bounded by L, $|f''(x)| \leq L$ for every $x \in \mathbb{R}^d$. For multivariate twice-differentiable functions, it is equivalent to assuming that the singular values of the Hessian matrix $\nabla^2\, f(x)$ are upper bounded by L for every $x \in \mathbb{R}^d$. For the least squares problem (1), the individual Lipschitz constants L_i are given by $L_i = \|a_i\|^2$, while, for the L2-regularized logistic regression problem (3), we have $L_i = 0.25\|a_i\|^2 + \lambda$.

The second assumption we consider in this section lower bounds the curvature of the functions.

Assumption 2: The function f is μ-strongly convex, meaning that the function $x \mapsto f(x) - \frac{\mu}{2}\|x\|^2$ is convex for some $\mu > 0$. Furthermore, $f_i : \mathbb{R}^d \to \mathbb{R}$ is convex for each $i = 1, \ldots, n$.

This is a strong assumption. While each f_i is convex in the least squares problem (1), the overall function f is strongly convex if and only if the design matrix $A := [a_1, \ldots, a_n]$ has full row rank. On the other hand, the L2-regularized logistic regression problem (3) satisfies this assumption with $\mu \geq \lambda$ due to the presence of the regularizer. As we detail in Section IV, it is possible to relax the strong convexity assumption, as well as the assumption that each f_i is convex.

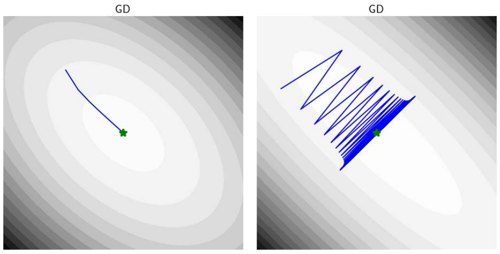

Fig. 3. *Here, we graph the level sets two logistic regression loss functions. The left level sets are each of a well-conditioned logistic function with $\kappa \approx 1$. The right level sets are of an ill-conditioned logistic function with $\kappa \gg 1$.*

An important problem class where the assumptions are satisfied is the problems of the form

$$x_\star \in \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ f(x) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(a_i^\top x) + \frac{\lambda}{2}\|x\|^2 \right\} \tag{15}$$

in the case when each "loss" function $\ell_i : \mathbb{R} \mapsto \mathbb{R}$ is twice differentiable with ℓ_i'' bounded between 0 and some upper bound M. This includes a variety of loss functions with L2-regularization in machine learning, such as least squares $(l_i(\alpha) = (\alpha - b_i)^2)$, logistic regression, probit regression, the Huber robust regression, and a variety of others. In this setting, for all i, we have $L_i \leq M\|a_i\|^2 + \lambda$ and $\mu \geq \lambda$.

The convergence rate of GD under these assumptions is determined by the ratio $\kappa := L/\mu$, which is known as the condition number of f. This ratio is always greater than or equal to 1, and when it is significantly larger than 1, the level sets of the function become very elliptical, which causes the iterates of the GD method to oscillate. This is illustrated in Fig. 3. In contrast, when κ is close to 1, GD converges quickly.

Under Assumptions 1 and 2, VR methods converge at a linear rate. We say that the function values $\{f(x_k)\}$ of a randomized method converge linearly (in expectation) at a rate of $0 < \rho \leq 1$ if there exists a constant $C > 0$ such that

$$\mathbf{E}\left[f(x_k)\right] - f(x_\star) \leq (1-\rho)^k C = \mathcal{O}(\exp(-k\rho)) \quad \forall k. \tag{16}$$

This is in contrast to classic SGD methods that only rely on an unbiased estimate of the gradient in each iteration, which, under these assumptions, can only obtain the sublinear rate

$$\mathbf{E}\left[f(x_k)\right] - f(x_\star) \leq \mathcal{O}(1/k).$$

Thus, classic SGD methods become slower the longer we run them, while VR methods continue to cut the error by at least a fixed fraction in each step.

As a consequence of (16), we can determine the number of iterations needed to reach a given tolerance $\varepsilon > 0$ on the error as follows:

$$k \geq \frac{1}{\rho}\log\left(\frac{C}{\varepsilon}\right), \quad \text{then } \mathbf{E}\left[f(x_k)\right] - f(x_\star) \leq \varepsilon. \tag{17}$$

The smallest k satisfying this inequality is known as the iteration complexity of the algorithm. In the following, we give the iteration complexity and the cost of one iteration in terms of n for the basic variant of GD, SGD, and VR methods:

Algorithm	# Iterations	Cost of 1 Iteration
GD	$\mathcal{O}(\kappa\log(1/\varepsilon))$	$\mathcal{O}(n)$
SGD	$\mathcal{O}(\kappa_{\max}(1/\varepsilon))$	$\mathcal{O}(1)$
VR	$\mathcal{O}((\kappa_{\max}+n)\log(1/\varepsilon))$	$\mathcal{O}(1)$

The total runtime of an algorithm is given by the product of the iteration complexity and the iteration runtime. Above, we have used $\kappa_{\max} := (\max_i L_i)/\mu$. Note that $\kappa_{\max} \geq \kappa$; thus, the iteration complexity of GD is smaller than that of the VR methods.[4] However, the VR methods are superior in terms of total runtime since each iteration of GD costs n times more than an iteration of a VR method.[5] Classic SGD methods have the advantage that their runtime and their convergence rate do not depend on n, but it does have a much worse dependence on the tolerance ε, which explains SGD's poor performance when the tolerance is small.[6]

In Appendix B, we give a simple proof showing that the SGD$_\star$ method has the same iteration complexity as the VR methods.

II. BASIC VARIANCE-REDUCED METHODS

The first wave of VR methods that achieve the convergence rate from the previous section started with the SAG method [37], [65]. This was followed shortly after by the stochastic dual coordinate ascent (SDCA) [63], [70], MISO [44], stochastic VR gradient (SVRG/S2GD) [28], [32], [43], [84], and SAGA (SAG "amélioré") [13] methods. In this section, we present several of these original methods, while Section IV covers more recent methods that offer improved properties in certain settings over these original methods.

A. Stochastic Average Gradient

The first VR method is based on mimicking the structure of the full gradient. Since the full gradient $\nabla f(x)$ is a plain average of the $\nabla f_i(x)$ gradients, our estimate g_k of the full gradient should be an average of estimates of the $\nabla f_i(x)$ gradients. This idea leads us to our first VR method: the SAG method.

The SAG method [37], [65] is a stochastic variant of the earlier incremental aggregated gradient (IAG) method [4]. The idea behind SAG is to maintain an estimate $v_k^i \approx \nabla f_i(x_k)$ for each data point i. We then use the average of the v_k^i values as our estimate of the full gradient, that is,

$$\bar{g}_k = \frac{1}{n}\sum_{j=1}^{n} v_k^j \approx \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x_k) = \nabla f(x_k). \tag{18}$$

At each iteration, SAG samples $i_k \in \{1,\ldots,n\}$ and updates v_k^j using

$$v_{k+1}^j = \begin{cases}\nabla f_{i_k}(x_k), & \text{if } j = i_k\\ v_k^j, & \text{if } j \neq i_k\end{cases} \tag{19}$$

where each v_i^0 might be initialized to zero or to an approximation of $\nabla f_i(x_0)$. As we approach a solution $x_\star$, each v^i converges to $\nabla f_i(x_\star)$, which gives us the VR property (12).

To implement SAG efficiently, we need to take care in computing $\bar{g}_k$ using (18) since this requires summing up n vectors in $\mathbb{R}^d$, and since n can be very large, computing this sum can be very costly. Fortunately, we can avoid computing this summation from scratch every iteration since only one v_k^i term will change in the next iteration. That is, suppose that we sample the index i_k on iteration k. It follows from (18) and (19) that

$$\begin{aligned}\bar{g}_k &= \frac{1}{n}\sum_{j=1, j\neq i_k}^{n} v_k^j + \frac{1}{n}v_k^{i_k}\\ &= \frac{1}{n}\sum_{j=1, j\neq i_k}^{n} v_{k-1}^j + \frac{1}{n}v_k^{i_k} \quad (\text{Since } v_{k-1}^j = v_k^j \text{ for all } j \neq i_k)\\ &= \bar{g}_{k-1} - \frac{1}{n}v_{k-1}^{i_k} + \frac{1}{n}v_k^{i_k} \quad (\text{Plus and minus } \tfrac{1}{n}v_{k-1}^{i_k}).\end{aligned} \tag{20}$$

Since v_{k-1}^j are simply copied over to v_k^j, when implementing SAG, we can simply store one vector v^j for each j. This implementation is illustrated in Algorithm 1.

The SAG method was the first stochastic method to enjoy linear convergence with an iteration complexity of $\mathcal{O}((\kappa_{\max}+n)\log(1/\varepsilon))$, using a stepsize of $\gamma = \mathcal{O}(1/L_{\max})$. This linear convergence can be seen in Fig. 1. Note that since an $L_{\max}$-smooth function is also L'-smooth for any $L' \geq L_{\max}$, this method obtains a linear convergence rate for any sufficiently small stepsize. This is in contrast to classic SGD methods, which only obtains sublinear rates and only under difficult-to-tune-in-practice decreasing stepsize sequences.

At the time, the linear convergence of SAG was a remarkable breakthrough given that SAG only computes a single stochastic gradient (processing a single data point) at each iteration. However, the convergence proof by

[4] In Section IV, we discuss how nonuniform sampling within VR methods leads to a faster rate, depending on the mean $\bar{L} := (1/n)\sum_i L_i$ rather than on the maximum $\max_i L_i$.

[5] Since $L_{\max} \leq nL$.

[6] We have omitted an additional term for the SGD iteration complexity of the form $\mathcal{O}(\sigma^2/\mu\varepsilon)$, where $\sigma^2 \geq \mathbf{E}_i[\|\nabla f_i(x_*)\|^2]$ (see [49, Th. 2.1]).

Algorithm 1 SAG Method

```
Parameters: stepsize $\gamma > 0$
Initialize: $x_0, v^i = 0 \in \mathbb{R}^d$ for $i = 1, \ldots, n$
for $k = 1, \ldots, T-1$ do
    Sample $i_k \in \{1, \ldots, n\}$
    $\bar{g}_k = \bar{g}_{k-1} - \frac{1}{n} v^{i_k}$
    $v^{i_k} = \nabla f_{i_k}(x_k)$
    $\bar{g}_k = \bar{g}_k + \frac{1}{n} v^{i_k}$
    $x_{k+1} = x_k - \gamma \bar{g}_k$
Output: $x_T$
```

Schmidt *et al.* [65] is notoriously difficult and relies on computer-verified steps. What specifically makes SAG hard to analyze is that $\bar{g}_k$ is a biased estimate of the gradient. Next, we introduce the SAGA method, a variant of SAG that uses the concept of covariates to make an unbiased variant of the SAG method that has similar performance but is easier to analyze.

B. SAGA

A common way to reduce the variance of the basic unbiased estimate $\nabla f_{i_k}(x_k)$ is by using what is known as covariates (or "control variates"). Let $v^i \in \mathbb{R}^d$ be a vector for $i = 1, \ldots, n$. Using these vectors, we can rewrite our full gradient as

$$\begin{aligned}\nabla f(x) &= \frac{1}{n}\sum_{i=1}^{n}(\nabla f_i(x) - v^i + v^i)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\nabla f_i(x) - v^i + \frac{1}{n}\sum_{j=1}^{n} v^j\right)\\ &:= \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x, v)\end{aligned} \tag{21}$$

where $\nabla f_i(x, v) := \nabla f_i(x) - v^i + \frac{1}{n}\sum_{j=1}^{n} v^j$. Now, we can build an unbiased estimate of the full gradient $\nabla f(x)$ by sampling a single $\nabla f_i(x, v)$ uniformly for $i \in \{1, \ldots, n\}$. That is, we can solve (2) by applying the SGD method with the gradient estimate

$$g_k = \nabla f_{i_k}(x_k, v) = \nabla f_{i_k}(x_k) - v^i + \frac{1}{n}\sum_{j=1}^{n} v^j. \tag{22}$$

To see the effect of the choice of the v^i's on the variance of g_k, substituting $g_k = \nabla f_{i_k}(x_k, v)$ and using $\mathbf{E}_{i\sim\frac{1}{n}}[v^i] = (1/n)\sum_{j=1}^{n} v^j$ in (12) give

$$\begin{aligned}(12) &= \mathbf{E}\left[\|\nabla f_i(x_k) - v^i + \mathbf{E}_{i\sim\frac{1}{n}}[v^i - \nabla f_i(x_k)]\|^2\right]\\ &\leq \mathbf{E}\left[\|\nabla f_i(x_k) - v^i\|^2\right]\end{aligned} \tag{23}$$

where we used Lemma 2 with $X = \nabla f_i(x_k) - v^i$. This bound (23) shows us that we obtain the VR property (12) if v^i approaches $\nabla f_i(x_k)$ as k grows. This is why we refer to the v^i's as covariates. We are free to choose any v^i, so we can choose them to reduce the variance.

As an example, the SGD$_\star$ method (13) also implements this approach with $v^i = \nabla f_i(x_\star)$. However, again, this is not practical since often we do not know $\nabla f_i(x_\star)$. A more practical choice for v^i is the gradient $\nabla f_i(\bar{x}_i)$ around a point $\bar{x}_i \in \mathbb{R}^d$ that we do know. SAGA uses a reference point $\bar{x}_i \in \mathbb{R}^d$ for each function f_i and uses the covariate $v^i = \nabla f_i(\bar{x}_i)$ where each $\bar{x}_i$ will be the last point for which we evaluated $\nabla f_i(\bar{x}_i)$. Using these covariates, we can build a gradient estimate following (22), which gives

$$g_k = \nabla f_{i_k}(x_k) - \nabla f_{i_k}(\bar{x}_{i_k}) + \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(\bar{x}_j). \tag{24}$$

To implement SAGA, instead of storing the n reference points $\bar{x}_i$, we can store the gradients $\nabla f_i(\bar{x}_i)$. That is, let $v^j = \nabla f_j(\bar{x}_j)$ for $j \in \{1, \ldots, n\}$, and similar to SAG, we update v^j of one random gradient in each iteration. We formalize the SAGA method as Algorithm 2, which is similar to the implementation of SAG (see Algorithm 1); except now, we store the previously known gradient of f_{i_k} in a dummy variable v^{old} so that we can then form the unbiased gradient estimate (24).

Algorithm 2 SAGA

```
Parameters: stepsize $\gamma > 0$
Initialize: $x_0, v^i = 0 \in \mathbb{R}^d$ for $i = 1, \ldots, n$
for $k = 1, \ldots, T-1$ do
    Sample $i_k \in \{1, \ldots, n\}$
    $v^{\text{old}} = v^{i_k}$
    $v^{i_k} = \nabla f_{i_k}(x_k)$
    $x_{k+1} = x_k - \gamma\left(v^{i_k} - v^{\text{old}} + \bar{g}_k\right)$
    $\bar{g}_k = \bar{g}_{k-1} + \frac{1}{n} v^{i_k} - \frac{1}{n} v^{\text{old}}$
Output: $x_T$
```

The SAGA method has an iteration complexity of $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$ using a stepsize of $\gamma = \mathcal{O}(1/L_{\max})$ as in SAG but with a much simpler proof. However, as with SAG, the SAGA method needs to store the auxiliary vectors $v^i \in \mathbb{R}^d$ for $i = 1, \ldots, n$, which amounts to an $\mathcal{O}(nd)$ storage. This can be infeasible when both d and n are large. We detail, in Section III, how we can reduce this memory requirement for common models, such as regularized linear models (15).

When the n auxiliary vectors can be stored in memory, SAG and SAGA tend to perform similarly. When this memory requirement is too high, the SVRG method that we review next is a good alternative. The SVRG method achieves the same convergence rate and is often nearly as

Algorithm 3 SVRG Method

```
1: Parameters stepsize $\gamma > 0$
2: Initialization $\bar{x}_0 = x_0 \in \mathbb{R}^d$
3: for $s = 1, 2, \ldots$ do
4:     Compute and store $\nabla f(\bar{x}_{s-1})$
5:     $x_0 = \bar{x}_{s-1}$
6:     Choose the number of inner-loop iterations $t$
7:     for $k = 0, 1, \ldots, t-1$ do
8:         Sample $i_k \in \{1, \ldots, n\}$
9:         $g_k = \nabla f_{i_k}(x_k) - \nabla f_{i_k}(\bar{x}_{s-1}) + \nabla f(\bar{x}_{s-1})$
10:        $x_{k+1} = x_k - \gamma g_k$
11:    $\bar{x}_s = x_t$.
```

fast in practice but only requires $\mathcal{O}(d)$ memory for general problems.

C. SVRG

Prior to SAGA, the first works to use covariates used them to address the high memory required of SAG [28], [43], [84]. These works build covariates based on a fixed reference point $\bar{x} \in \mathbb{R}^d$ at which we have already computed the full gradient $\nabla f(\bar{x})$. By storing $\bar{x}$ and $\nabla f(\bar{x})$, we can implement the update (24) using $\bar{x}_j = \bar{x}$ for all j without storing the individual gradients $\nabla f_j(\bar{x})$. In particular, instead of storing these vectors, we compute $\nabla f_{i_k}(\bar{x})$ in each iteration using the stored reference point $\bar{x}$. Originally presented under different names by different authors, this method has come to be known as the SVRG method, following the naming of [28] and [84].

We formalize the SVRG method in Algorithm 3. Using (23), we have that the variance of the gradient estimate g_k is bounded by

$$\begin{aligned} \mathbf{E}\left[\|g_k - \nabla f(x_k)\|^2\right] &\le \mathbf{E}\left[\|\nabla f_i(x_k) - \nabla f_i(\bar{x})\|^2\right] \\ &\le L_{\max}^2 \|x_k - \bar{x}\|^2 \end{aligned}$$

where the second inequality uses the L_i-smoothness of each L_i.[7] Notice that the closer $\bar{x}$ is to x_k, the smaller the variance of the gradient estimate.

To make the SVRG method work well, we need to trade off the cost of updating the reference point $\bar{x}$ frequently and, thus, having to compute the full gradient, with the benefits of decreasing the variance. To do this, the reference point is updated every t iterations to be a point close to x_k (see line 11 of Algorithm II-C). That is, the SVRG method has two loops: one outer loop in s where the reference gradient $\nabla f(\bar{x}_{s-1})$ is computed (line 4), and one inner loop where the reference point is fixed, and the inner iterates x_k are updated (line 10) according to stochastic gradient steps using (22).

In contrast to SAG and SAGA, SVRG requires $\mathcal{O}(d)$ memory only. The downsides of SVRG are: 1) we have an additional parameter t, the length of the inner loop, which needs to be tuned and 2) two gradients are computed per iteration and the full gradient needs to be computed every time the reference point is changed.

Johnson and Zhang [28] showed that SVRG has iteration complexity $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$, similar to SAG and SAGA. This was shown assuming that the number of inner iterations t is sampled uniformly from $\{1, \ldots, m\}$, where a complex dependence must hold between $L_{\max}$, μ, the stepsize γ, and t. In practice, SVRG tends to work well by using $\gamma = \mathcal{O}(1/L_{\max})$ and inner loop length $t = n$, which, is the setting, we used in Fig. 1.

There are, now, many variations on the original SVRG method. For example, there are variants that use alternative distributions for t [32] and variants that allow stepsizes of the form $\mathcal{O}(1/L_{\max})$ [27], [33], [35]. There are also variants that use a minibatch approximation of $\nabla f(\bar{x})$ to reduce the cost of these full-gradient evaluations and that grow the minibatch size in order to maintain the VR property [17], [26]. There are variants that repeatedly update g_k in the inner loop according to [54]

$$g_k = \nabla f_{i_k}(x_k) - \nabla f_{i_k}(x_{k-1}) + g_{k-1} \qquad (25)$$

which provides a more local approximation. Using this continuous update variant (25) has shown to have distinct advantages in minimizing nonconvex functions, as we briefly discuss in Section IV-G. Finally, note that SVRG can use the values of $\nabla f(\bar{x}_s)$ to help decide when to terminate the algorithm.

D. SDCA and Variants

A drawback of SAG and SVRG is that their stepsize depends on $L_{\max}$, which may not be known for some problems. One of the first VR methods (predating SVRG) was the SDCA method [70] that extended recent work on coordinate descent methods to the finite-sum problem.[8]

The intuition behind SDCA, and its variants, is that the coordinates of the gradient provide a naturally VR estimate of the gradient. That is, let $j \in \{1, \ldots, d\}$ and $\nabla_j f(x) := (\partial f(x)/\partial x_j) e_j$ be the coordinatewise derivative of $f(x)$, where $e_j \in \mathbb{R}^d$ is the jth unit coordinate vector. An important feature of coordinatewise derivatives is that $\nabla_j f(x_\star) = 0$ since we know that $\nabla f(x_\star) = 0$. This is unlike the derivative for each data point ∇f_j that may be different

[7] When each f_i is also convex, we can derive the bound $\mathbb{E}[\|\nabla f(\bar{x}_{s-1}) - \nabla f(x_k)\|^2] \le 4L_{\max}(f(x_k) - f(x^*) + f(\bar{x}_{s-1}) - f(x^*))$ using analogous proof to Lemma 1. This bound on the variance is key to proving a good convergence rate for SVRG in the convex setting.

[8] The modern interest in coordinate ascent/descent methods began with [51], which considered coordinatewise GD with randomly chosen coordinates and included a result showing linear convergence for L-smooth strongly convex functions. This led to an explosion of work on the problem; as for many problem structures, we can very efficiently compute coordinatewise GD steps [63]

than zero at $x_\star$. Due to this feature, we have that

$$\|\nabla f(x) - \nabla_j f(x)\|^2 \underset{x\to x_\star}{\longrightarrow} 0. \tag{26}$$

Thus, coordinatewise derivative satisfies the VR property (12). Furthermore, we can also use $\nabla_j f(x)$ to build an unbiased estimate of $\nabla f(x)$. For instance, let j be a random index sampled uniformly on average from $\{1,\dots,d\}$. Thus, for any given $i \in \{1,\dots,d\}$, we have that $\mathbf{P}[j=i] = (1/d)$. Consequently, $d \times \nabla_j f(x)$ is an unbiased estimate of $\nabla f(x)$ since

$$\begin{aligned}\mathbf{E}[d\nabla_j f(x)] &= d\sum_{i=1}^{d}\mathbf{P}[j=i]\frac{\partial f(x)}{\partial x_i}e_i = \sum_{i=1}^{d}\frac{\partial f(x)}{\partial x_i}e_i\\ &= \nabla f(x).\end{aligned}$$

Thus, $\nabla_j f(x)$ has all the favorable properties that we would like for a VR estimate of the full gradient without using covariates. The downside of using this coordinatewise gradient is that, for our sum-of-terms problem (2), it is expensive to compute. This is because computing $\nabla_j f(x)$ requires a full pass over the data since

$$\nabla_j f(x) = \frac{1}{n}\sum_{i=1}^{n}\nabla_j f_i(x).$$

Thus, it would seem that using coordinatewise derivatives is incompatible with the structure of our sum-of-terms problem. Fortunately though, we can often rewrite our original problem (2) in what is known as a dual formulation where coordinatewise derivatives can make use of the inherent structure.

To illustrate, the dual formulation of the L2-regularized linear models of the form (15) is given by

$$v^\star \in \underset{v\in\mathbb{R}^n}{\operatorname{argmax}}\left\{\frac{1}{n}\sum_{i=1}^{n}-\ell_i^\star(-v^i) - \frac{\lambda}{2}\|\frac{1}{\lambda n}\sum_{i=1}^{n}v^i a_i\|^2\right\} \tag{27}$$

where $\ell_i^\star(v) := \sup_x\{\langle x,v\rangle - f(x)\}$ is the convex conjugate of ℓ_i. We can recover the x variable of our original problem (15) using the mapping

$$x = \frac{1}{\lambda n}\sum_{i=1}^{n}v^i a_i.$$

Consequently, plugging in the solution $v^\star$ to (27) in the right-hand side of the above gives $x_\star$, the solution of (15).

Notice that this dual problem has n real variables $v^i \in \mathbb{R}$, one for each training example. Furthermore, each dual loss function $\ell_i^\star$ in (27) is a function of a single v^i only. That is, the first term in the loss function is separable over coordinates. It is this separability over coordinates, combined with the simple form of the second term, that allows for an efficient implementation of a coordinate ascent method.[9] Indeed, Shalev-Shwartz and Zhang [70] showed that coordinate ascent on this dual problem has an iteration complexity of $\mathcal{O}((\kappa_{\max}+n)\log(1/\varepsilon))$, similar to SAG, SAGA, and SVRG.[10] The iteration cost and algorithm structure are also quite similar: by keeping track of the sum $\sum_{i=1}^{n} v^i a_i$ to address the second term in (27), each dual coordinate ascent iteration only needs to consider a single training example, and the cost per iteration is independent of n. Furthermore, we can use a 1-D line search to efficiently compute a stepsize that maximally increases the dual objective as a function of one v^i. This means that the fast worst case runtime of VR methods can be achieved with no knowledge of $L_{\max}$ or related quantities.

Unfortunately, the SDCA method also has several disadvantages. First, it requires computing the convex conjugates $\ell_i^\star$ rather than simply gradients. We do not have an equivalent of automatic differentiation for convex conjugates, so this may increase the implementation effort. More recent works have presented "dual-free" SDCA methods that do not require the conjugates and instead work with gradients [69]. However, it is no longer possible to track the dual objective in order to set the stepsize in these methods. Second, while SDCA only requires $\mathcal{O}(n+d)$ memory for problem (15), SAG/SAGA also only requires $\mathcal{O}(n+d)$ memory for this problem class (see Section III). Variants of SDCA that applies to more general problems have the $\mathcal{O}(nd)$ memory of SAG/SAGA since the v^i become vectors with d-elements. A final subtle disadvantage of SDCA is that it implicitly assumes that the strong convexity constant μ is equal to λ. For problems where μ is greater than λ, the primal VR methods often significantly outperform SDCA.

III. PRACTICAL CONSIDERATIONS

In order to implement the basic VR methods and to obtain a reasonable performance, several implementation issues must be addressed. In this section, we discuss several issues that are not addressed above.

A. Setting the Step Size for SAG/SAGA/SVRG

While we can naturally use the dual objective to set the stepsize for SDCA, the theory for the primal VR methods SAG/SAGA/SVRG relies on stepsizes of the form $\gamma = \mathcal{O}(1/L_{\max})$. Yet, in practice, one may not know $L_{\max}$, and better performance can often be obtained with other stepsizes.

One classic strategy for setting the stepsize in full-GD methods is the *Armijo line search* [3]. Given a current point x_k and a search direction g_k, the Armijo line search for a

[9] We call it "coordinate ascent" instead of "coordinate descent" since (27) is a maximization problem.

[10] This iteration complexity is in terms of the duality gap. Related results for certain problem structures include [10] and [74].

γ_k is on the line $\gamma_k \in \{\gamma \,:\, x_k + \gamma g_k\}$ and gives a sufficient decrease of the function

$$f(x_k + \gamma_k g_k) < f(x_k) - c\gamma_k \|\nabla f(x_k)\|^2. \tag{28}$$

This requires calculating $f(x_k + \gamma_k g_k)$ on several candidate stepsizes γ_k, which is prohibitively expensive since evaluating $f(x)$ requires a full pass over the data.

Thus, instead of using the full function $f(x)$, we can use a stochastic variant where we look for γ_k such that

$$f_{i_k}(x_k + \gamma_k g_k) < f_{i_k}(x_k) - c\gamma_k \|\nabla f_{i_k}(x_k)\|^2. \tag{29}$$

This is used in the implementation of [65] with $c = (1/2)$ on iterations where $\|\nabla f_{i_k}(x_k)\|$ is not close to zero. It often works well in practice with appropriate guesses for the trial stepsizes although no theory exists for the method.

Alternatively, Mairal [44] considers the "Bottou trick" for setting the stepsize in practice.[11] This method takes a small sample of the data set (typically 5%) and performs a binary search that attempts to find the optimal stepsize when performing one pass through this sample. Similar to the Armijo line search, the stepsize obtained with this method tends to work well in practice, but no theory is known for the method.

B. Termination Criteria

Iteration complexity results provide theoretical worst case bounds on the number of iterations to reach a certain accuracy. However, these bounds depend on constants that we may not know, and in practice, the algorithms tend to require fewer iterations than indicated by the bounds. Thus, we should consider tests to decide when the algorithm should be terminated.

In classic full-GD methods, we typically consider the norm of the gradient $\|\nabla f(x_k)\|$ or some variation on this quantity to decide when to stop. We can naturally implement these same criteria to decide when to stop an SVRG method, by using $\|\nabla f(\bar{x}_s)\|$. For SAG/SAGA, we do not explicitly compute any full gradients, but the quantity $\bar{g}_k$ converges to $\nabla f(x_k)$, so a reasonable heuristic is to use $\|\bar{g}_k\|$ in deciding when to stop. In the case of SDCA, with a small amount of extra bookkeeping, it is possible to track the gradient of the dual objective at no additional asymptotic cost. Alternately, a more principled approach is to track the duality gap, which adds an $\mathcal{O}(n)$ cost per iteration but leads to termination criteria with a duality gap certificate of optimality. An alternative principled approach based on optimality conditions for strongly convex objectives is used in the MISO method [44] based on a quadratic lower bound [41].

C. Reducing Memory Requirement

Although SVRG removes the memory requirement of earlier VR methods, in practice, SAG/SAGA requires fewer iterations than SVRG on many problems. Thus, we might consider whether there exist problems where SAG/SAGA can be implemented with less than $\mathcal{O}(nd)$ memory. In this section, we consider the class of linear models, where the memory requirement can be reduced substantially.

Consider linear models where $f_i(x) = \ell_i(a_i^\top x)$. Differentiating gives

$$\nabla f_i(x) = \ell_i'(a_i^\top x)a_i.$$

Provided we already have access to the feature vectors a_i, it is sufficient to store the scalars $\ell_i(a_i^\top x)$ in order to implement the SAG/SAGA method. This reduces the memory requirements from $\mathcal{O}(nd)$ down to $\mathcal{O}(n)$. SVRG can also benefit from this structure of the gradients: by storing those n scalars, we can reduce the number of gradient evaluations required per SVRG "inner" iteration to 1 for this problem class.

There exist other problem classes, such as probabilistic graphical models, where it is possible to reduce the memory requirements [66].

D. Sparse Gradients

For problems where the gradients $\nabla f_i(x)$ have many zero values (e.g., for linear models with sparse features), the classical SGD update may be implemented with complexity, which is linear in the number of nonzero components in the corresponding gradient, which is often much less than d. This possibility is lost in plain VR methods. However, there are two known fixes.

The first one, described by Schmidt *et al.* [65, Sec. 4.1], takes advantage of the simple form of the updates to implement a "just-in-time" variant where the iteration cost is proportional to the number of nonzeroes. For SAG (but this applies to all variants), this is done by not explicitly storing the full vector v^{i_k} after each iteration. Instead, in each iteration, we only compute the elements $v_j^{i_k}$ corresponding to nonzero elements, by applying the sequence of updates to each variable $v_j^{i_k}$ since the last iteration where it was nonzero.

The second one, described by Leblond *et al.* [38, Section 2] for SAGA, adds an extra randomness to the update $x_{k+1} = x_k - \gamma(\nabla f_{i_k}(x_k) - \nabla f_{i_k}(\bar{x}_{i_k}) + \bar{g}_k)$, where $\nabla f_{i_k}(x_k)$ and $\nabla f_{i_k}(\bar{x}_{i_k})$ are sparse, but $\bar{g}_k$ is dense. The components of the dense term $(\bar{g}_k)_j$, $j = 1, \ldots, d$, are replaced by $w_j(\bar{g}_k)_j$, where $w \in \mathbb{R}^d$ is a random sparse vector whose support is included in one of the $\nabla f_{i_k}(x_k)$ and, in expectation, is the constant vector of all ones. The update remains unbiased (but is now sparse), and the added variance does not impact the convergence rate; the details are given by Leblond *et al.* [38].

[11] Introduced publicly by Léon Bottou during his tutorial on SGD methods at NeurIPS 2017.

IV. ADVANCED ALGORITHMS

In this section, we consider extensions of the basic VR methods. Some of these extensions generalize the basic methods to handle more general scenarios, such as problems that are not smooth and/or strongly convex. Other extensions use additional algorithmic tricks or problem structure to design faster algorithms than the basic methods.

A. Hybrid SGD and VR Methods

The convergence rate ρ of the VR methods depends on the number of training examples n. This is in contrast to the convergence rates of classic SGD methods that are sublinear but do not have a dependence on n. This means that VR methods can perform worse than classic SGD methods in the early iterations when n is very large. For example, in Fig. 1, we can see that SGD is competitive with the two VR methods throughout the first 10 epochs (passes over the data).

Several hybrid SGD and VR methods have been proposed to improve the dependence of VR methods on n. Konečný and Richtárik [32] and Le Roux *et al.* [37] analyzed SAG and SVRG, respectively, when initialized with n iterations of SGD. This does not change the convergence rate but significantly improves the dependence on n in the constant factor. However, this requires setting the stepsize for these initial SGD iterations, which is more complicated than setting the stepsize for VR methods.[12]

More recently, several methods have been explored, which guarantee both a linear convergence rate depending on n, as well as a sublinear convergence rate that does not depend on n. For example, Lei and Jordan [39] show that this "best of both worlds" result can be achieved for the "practical" SVRG variant where we use a growing minibatch approximation of $\nabla f(\bar{x})$.

B. Nonuniform Sampling

Instead of improving the dependence on n, a variety of works have focused on improving the dependence on the Lipschitz constants L_i by using nonuniform sampling of the random training example i_k. In particular, these algorithms bias the choice of i_k toward the larger L_i values. This means that examples whose gradients can change more quickly are sampled more often. This is typically combined with using a larger stepsize that depends on the average of the L_i values rather than the maximum L_i value. Under an appropriate choice of the sampling probabilities and stepsize, this leads to improved iteration complexities of the form

$$\mathcal{O}((\kappa_{\text{mean}} + n)\log(1/\varepsilon))$$

which depends on $\kappa_{\text{mean}} := ((1/n)\sum_i L_i)/\mu$ rather than on $\kappa_{\max} = (\max_i L_i)/\mu$. This improved rate under nonuniform sampling has been shown for the basic VR methods SVRG [81], SDCA [60], and SAGA [25], [66].

Virtually, all existing methods use fixed probability distribution over $\{1,\ldots,n\}$ throughout the iterative process. However, it is possible to further improve on this choice by adaptively changing the probabilities during the execution of the algorithm. The first VR method of this type, ASDCA, was developed by Csiba *et al.* [12] and is based on updating the probabilities in SDCA by using what is known as the dual residue.

Schmidt *et al.* [65] present an empirical method that tries to estimate local values of the L_i (which may be arbitrarily smaller than the global values) and show impressive gains in experiments. A related method with a theoretical backing that uses local L_i estimates is presented by Vainsencher *et al.* [76].

C. Minibatching

Another strategy to improve the dependence on the L_i values is to use minibatching, analogous to the classic minibatch SGD method (7), to obtain a better approximation of the gradient. There are a number of ways of doing minibatching, but here we will focus on a fixed batch-size chosen uniformly at random. That is, let $b \in \mathbb{N}$, and we choose a set $B_k \subset \{1,\ldots,n\}$ with $|B_k| = b$ with uniform probability from all sets with b elements. We can now implement the VR method by replacing each $\nabla f_i(x^k)$ with a minibatch estimate given by $(1/b)\sum_{i\in B_k}\nabla f_i(x^k)$.

There were a variety of early minibatch methods [26], [27], [31], [75], but the most recent methods are able to obtain an iteration complexity of the form [19], [58], [60], [68]

$$\mathcal{O}\left(\frac{\mathcal{L}(b)}{\mu} + \frac{n}{b}\right)\log\left(\frac{1}{\varepsilon}\right) \tag{30}$$

using a stepsize of $\gamma = \mathcal{O}(1/\mathcal{L}(b))$, where

$$\mathcal{L}(b) = \frac{1}{b}\frac{n-b}{n-1}L_{\max} + \frac{n}{b}\frac{b-1}{n-1}L \tag{31}$$

is a minibatch smoothness constant first defined in [24] and [25]. This iteration complexity interpolates between the complexity of full-GD when $\mathcal{L}(n) = L$ and VR methods where $\mathcal{L}(1) = L_{\max}$. Since $L \le L_{\max} \le nL$, it is possible that $L \lll L_{\max}$. Thus, using larger minibatches allows for the possibility of large speedups, especially in settings where we can use parallel computing to evaluate multiple gradients simultaneously. However, computing L is typically more challenging than computing $L_{\max}$.

For generalized linear models (15) with $\ell'' < M$, we have that $L \le M\lambda_{\max}(AA^\top)$, where $A = [a_1,\ldots,a_n] \in \mathbb{R}^{d\times n}$ and $\lambda_{\max}(\cdot)$ is the largest eigenvalue function. The largest eigenvalue can be computed using a reduced SVD

[12] The implementation of Schmidt *et al.* [65] does not use this trick. Instead, it replaces n in line 7 of Algorithm II-B with the number of training examples that have been sampled at least once. This leads to similar performance and is more difficult to analyze but avoids needing to tune an additional stepsize.

at a cost of $\mathcal{O}(d^2\ n)$ or by using a few iterations of the power method to get an approximation. Unfortunately, for some problems, this cost may negate the advantage of using minibatches. In such a case, we could replace L with the upper bound $(1/n)\sum_{i=1}^n L_i$. However, this may be a very conservative upper bound.

D. Accelerated Variants

An alternative strategy for improving the dependence on $\kappa_{\max}$ is Nesterov or Polyak acceleration (also known as momentum). It is well known that Nesterov's accelerated GD improves the iteration complexity of the full-gradient method from $\mathcal{O}(\kappa \log(1/\varepsilon))$ to $\mathcal{O}(\sqrt{\kappa}\log(1/\varepsilon))$ [50]. Although we might naively hope to see the same improvement for VR methods, replacing the $\kappa_{\max}$ dependence with $\sqrt{\kappa_{\max}}$, we now know that the best complexity that we can hope to achieve is $\mathcal{O}((\sqrt{n\kappa_{\max}}+n)\log(1/\varepsilon))$ [36], [80], which was first achieved by an accelerated SDCA method [71]. Nevertheless, this complexity still guarantees better worst case performance in ill-conditioned settings (where $\kappa_{\max} \gg n$).

A variety of VR methods have been proposed that incorporate an acceleration step in order to achieve this improved complexity (see [1], [71], and [85]). Moreover, the "catalyst" framework of Lin *et al.* [41] can be used to modify any method achieving the complexity $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$ to an accelerated method with a complexity of $\mathcal{O}((\sqrt{n\kappa_{\max}}+n)\log(1/\varepsilon))$.

E. Relaxing Smoothness

A variety of methods have been proposed that relax the assumption that f is L-smooth. The first of these was the SDCA method, which can still be applied to (2) even if the functions $\{f_i\}$ are nonsmooth. This is because the dual remains a smooth problem. A classic example is the SVM loss, where $f_i(x) = \max\{0, 1 - b_i a_i^\top x\}$. This leads to a convergence rate of the form $\mathcal{O}(1/\varepsilon)$ rather than $\mathcal{O}(\log(1/\varepsilon))$, so does not give a worst case advantage over classic SGD methods. However, unlike classic SGD methods, we can optimally set the stepsize when using SDCA. Indeed, prior to the new wave of VR methods, dual coordinate ascent methods have been among the most popular approaches for solving SVM problems for many years. For example, the widely popular libSVM package [7] uses a dual coordinate ascent method.

One of the first ways to handle nonsmooth problems that preserves the linear convergence rate was through the use of proximal-gradient methods. These methods apply when f has the form $f(x) = \frac{1}{}n\sum_{i=1}^n f_i(x) + \Omega(x)$. In this framework, it is assumed that f is L-smooth and that the regularizer Ω is convex on its domain. However, the function Ω may be nonsmooth and may enforce constraints on x. However, Ω must be "simple" in the sense that it is possible to efficiently compute its proximal operator applied to step of GD, that is,

$$x_{k+1} = \underset{x\in\mathbb{R}^d}{\operatorname{argmin}}\ \frac{1}{2}\|x - (x_k - \gamma\nabla f(x_k))\|^2 + \gamma\Omega(x) \quad (32)$$

should be relatively inexpensive to compute. The above method is known as the proximal-gradient algorithm [11, see, e.g.,], and it achieves the iteration complexity $\mathcal{O}(\kappa\log(1/\varepsilon))$ even though Ω (and consequently f) is not L-smooth or even necessarily differentiable. A common example where (32) can be efficiently computed is the L1-regularizer, where $\Omega(x) = \lambda\|x\|_1$ for some regularization parameter $\lambda > 0$.

A variety of works have shown analogous results for proximal variants of VR methods. These works essentially replace the iterate update by an update of the above form, with $\nabla f(x_k)$ replaced by the relevant approximation g_k. This leads to iteration complexities of $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$ for proximal variants of SAG/SAGA/SVRG if the functions f_i are L_i smooth [13], [81].

Several authors have also explored combinations of VR methods with the alternating direction method of multipliers (ADMM) approaches [86], which can achieve improved rates in some cases where Ω does not admit an efficient proximal operator. Several authors have also considered the case where the individual f_i may be nonsmooth, replacing them with a smooth approximation [1, see]. This smoothing approach does not lead to linear convergence rates but leads to faster sublinear rates that are obtained with SGD methods on nonsmooth problems (even with smoothing).

F. Relaxing Strong Convexity

While we have focused on the case where f is strongly convex and each f_i is convex, these assumptions can be relaxed. For example, if f is convex but not strongly convex, then early works showed that VR methods achieve a convergence rate of $\mathcal{O}(1/k)$ [13], [32], [43], [44], [65]. This is the same rate achieved by GD under these assumptions and is faster than the $\mathcal{O}(1/\sqrt{k})$ rate of SGD in this setting.

Alternatively, more recent works replace strong convexity with weakened assumptions, such as the PL-inequality and KL-inequality [57], which include standard problems, such as (unregularized) least squares, where it is still possible to show linear convergence [21], [29], [61], [62].

G. Nonconvex Problems

Since 2014, a sequence of articles have gradually relaxed the convexity assumptions on the functions f_i and f and adapted the variance reduction methods to achieve state-of-the-art complexity results for several different nonconvex settings. Here, we summarize some of these results, starting with the setting closest to the strongly convex setting and gradually relaxing any such convexity assumptions. For a more detailed discussion, see [15] and [87].

The first assumption that we relax is the convexity of the f_i functions. The first work in this direction was for solving the PCA problem where the individual f_i functions are nonconvex [18], [72]. Both Garber and Hazan [18] and Shalev-Shwartz [69] then showed how to use the catalyst framework [41] to devise algorithms with an iteration complexity of $\mathcal{O}(n+n^{3/4}\sqrt{L_{\max}}/\sqrt{\mu})\log(1/\varepsilon)$ for L_i–smooth nonconvex f_i functions so long as their average f is μ-strongly convex. Recently, this complexity was shown to match the lower bound in this setting [87]. Allen-Zhu [2] took a step further and relaxed the assumption that f is strongly convex and, instead, allowed f to be simply convex or even have "bounded nonconvexity" (strong convexity but with a negative parameter $-\mu$). To tackle this setting, Allen-Zhu [2] proposed an accelerated variant of SVRG that achieves state-of-the-art complexity results that recently have been shown to be optimal [87].[13]

Even more recently, the convexity assumptions on f have been completely dropped. By only assuming that f_i's are smooth and f has a lower bound, Fang *et al.* [15] present an algorithm based on the continuous update (25) that finds an approximate stationary point such that $\mathbf{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon$ using $\mathcal{O}(n+\sqrt{n}/\varepsilon^2)$ iterations. Concurrently to this, Zhou *et al.* [88] presented a more involved variant of SVRG that uses multiple reference points and achieves the same iteration complexity. Fang *et al.* [15] also provided a lower bound showing that the preceding complexity is optimal under these assumptions.

An interesting source of nonconvex functions is the problems where the objective is a composition of functions $f(g(x))$, where $g:\mathbb{R}^d \rightarrow \mathbb{R}^m$ is a mapping. Even when both f and g are convex, their composition may be nonconvex. There are several interesting applications where either g is itself an average (or even expectation) of maps or f is an average of functions, or even both [40]. In this setting, the finite sum structures can also be exploited to develop VR methods, most of which are based on variants of SVRG. In the setting where g is a finite sum, state-of-the-art complexity results have been achieved using the continuous update (25) (see [83]).

H. Second-Order Variants

Inspired by Newton's method, there are now second-order variants of the VR methods of the form

$$x_{k+1} = x_k - \gamma_k H_k g_k \tag{33}$$

where $H_k \in \mathbb{R}^{d\times d}$ is an estimate of the inverse Hessian matrix $\nabla^2 f(x_k)^{-1}$. The challenge in designing such methods is finding an efficient way to update H_k that results in a sufficiently accurate estimate and does not cost too much. This is a difficult balance to achieve; if H_k is a poor estimate, it may do more damage to the convergence of (33) than help. On the other hand, expensive routines for updating H_k can make the method inapplicable when the number of data points is large.

Though difficult, the rewards are potentially high. The second-order methods can be invariant (or at least insensitive) to coordinate transformations and ill-conditioned problems. This is in contrast to the first-order methods that often require some feature engineering, preconditioning, and data preprocessing to converge.

Most of the second-order VR methods are based on the BFGS quasi-Newton update [5], [16], [20], [73].

The first stochastic BFGS method that makes use of subsampling was the online L-BFGS method [67] that uses subsampled gradients to approximate Hessian-vector products. The regularized BFGS method [46], [47] also makes use of stochastic gradients and further modifies the BFGS update by adding a regularizer to the H_k matrix.

The first method to use subsampled Hessian-vector products in the BFGS update, as opposed to using differences of stochastic gradients, was the SQN method [6]. Moritz *et al.* [48] then proposed combining SQN with SVRG. The resulting method performs very well in numerical tests and was the first example of a second-order VR method with a proven linear convergence, though with a significantly worse complexity than the $\mathcal{O}(\kappa_{\max} + n)\log(1/\epsilon)$ rate of the VR methods. Moritz *et al.*'s [48] method was later extended to a block quasi-Newton variant and analyzed with an improved complexity by Gower *et al.* [23]. There are also specialized variants for the nonconvex setting [78]. These second-order quasi-Newton variants of the VR methods are hard to analyze, and as far as we are aware, there exist no quasi-Newton variants of (33), which has an update cost independent of n and is proved to have a better global complexity than $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$ of VR methods.

However, there do exist stochastic Newton-type methods, such as stochastic dual newton ascent (SDNA) [59], which performs minibatch-type Newton steps in the dual space, with a cost that is independent of n and does have a better convergence rate than SDCA. Furthermore, more recently, Kovalev *et al.* [34] proposed a minibatch Newton method with a local linear convergence rate of the form $\mathcal{O}((n/b)\log(1/\varepsilon))$ that is independent of the condition number, where b is the minibatch size. Yet another line of stochastic second-order methods is being developed by combining variance reduction techniques with the cubic regularized Newton method [53]. These methods replace both the Hessian and gradient by VR estimates and then minimize, at each iteration, an approximation of the second-order Taylor expansion with an additional cubic regularizer. What is particularly promising about these methods is that they achieve state-of-the-art sample complexity, in terms of accesses gradients and Hessians of individual functions f_i, for finding second-order stationary points for smooth nonconvex problems [79], [89]. This is currently a very active direction of research.

[13] Note that, when assuming that f has a bounded nonconvexity, the complexity results are with respect to finding a point $x_k \in \mathbb{R}^d$ such that $\mathbf{E}\left[\|\nabla f(x_k)\|^2\right] \leq \varepsilon$.

V. DISCUSSION AND LIMITATIONS

In Section IV, we presented a variety of extensions of the basic VR methods. We presented these as separate extensions, but many of them can be combined. For example, we can have an algorithm that uses minibatching and acceleration while using a proximal-gradient framework to address nonsmooth problems. The literature has now filled out most of these combinations.

Note that classic SGD methods apply to the general problem of minimizing a function of the form $f(x) = \mathbb{E}_z f(x,z)$ for some random variable z. In this work, we have focused on the case of the training error, where z can only take n values. However, in machine learning, we are ultimately interested in the test error where z may come from a continuous distribution. If we have an infinite source of examples, we can use them within SGD to directly optimize the test loss function. Alternatively, we can treat our n training examples as samples from the test distribution, and then, doing one pass of SGD through the training examples can be viewed as directly making progress on the test error. Although it is not possible to improve on the test error convergence rate by using VR methods, several works show that VR methods can improve the constants in the test error convergence rate compared with SGD [17], [26].

We have largely focused on the application of VR methods to linear models while mentioning several other important machine learning problems, such as graphical models and principal component analysis. One of the most important applications in machine learning where VR methods have had a little impact is in training deep neural networks. Indeed, recent work shows that VR may be ineffective at speeding up the training of deep neural networks [14]. On the other hand, VR methods are now finding applications in a variety of other machine learning applications, including policy evaluation for reinforcement learning [56], [77], [82], expectation maximization [9], simulations using Monte Carlo methods [90], saddle point problems [55], and generative adversarial networks [8].

APPENDIX A LEMMAS

Here, we provide and prove several auxiliary lemmas.

Lemma 1: Let $f_i(x)$ be convex and $L_{\max}$-smooth for $i = 1, \ldots, n$. Let i be sampled uniformly on average from $\{1, \ldots, n\}$. It follows that, for every $x \in \mathbb{R}^d$, we have that:

$$\mathbf{E}_k[\|\nabla f_i(x) - \nabla f_i(x_\star)\|^2] \;\leq\; 2L_{\max}(f(x) - f(x_\star)). \tag{34}$$

Proof: Since f_i is convex, we have that

$$f_i(z) \geq f_i(x_\star) + \langle \nabla f_i(x_\star), z - x_\star\rangle \quad \forall z \in \mathbb{R}^d \tag{35}$$

and since f_i is smooth we have that (see [52, Sec. 2.1.1] for more equivalent ways of re-writing convexity and smoothness)

$$f_i(z) \leq f_i(x) + \langle \nabla f_i(x), z - x\rangle + \frac{L_{\max}}{2}\|z - x\|^2 \quad \forall z, x \in \mathbb{R}^d. \tag{36}$$

It follows for all $x, z \in \mathbb{R}^d$ that:

$$\begin{aligned} f_i(x_\star) - f_i(x) &= f_i(x_\star) - f_i(z) + f_i(z) - f_i(x) \\ &\overset{(35)+(36)}{\leq} \langle \nabla f_i(x_\star), x_\star - z\rangle + \langle \nabla f_i(x), z - x\rangle \\ &\quad + \frac{L_{\max}}{2}\|z - x\|^2. \end{aligned}$$

To get the tightest upper bound on the right-hand side, we can minimize the right-hand side in z, which gives

$$z = x - \frac{1}{L_{\max}}(\nabla f_i(x) - \nabla f_i(x_\star)). \tag{37}$$

Substituting this in the above equation gives

$$\begin{aligned} &f_i(x_\star) - f_i(x) \\ &\quad = \left\langle \nabla f_i(x_\star), x_\star - x + \frac{1}{L_{\max}}(\nabla f_i(x) - \nabla f_i(x_\star))\right\rangle \\ &\qquad - \frac{1}{L_{\max}}\langle \nabla f_i(x), \nabla f_i(x) - \nabla f_i(x_\star)\rangle \\ &\qquad + \frac{1}{2L_{\max}}\|\nabla f_i(x) - \nabla f_i(x_\star)\|^2 \\ &\quad = \langle \nabla f_i(x_\star), x_\star - x\rangle \\ &\qquad - \frac{1}{2L_{\max}}\|\nabla f_i(x) - \nabla f_i(x_\star)\|^2. \end{aligned}$$

Taking expectation over i in the above and using that $\mathbf{E}_i[f_i(x)] = f(x)$ and $\mathbf{E}[\nabla f_i(x_\star)] = 0$ give the result. ■

Lemma 2: Let $X \in \mathbb{R}^d$ be a random vector with finite variance. It follows that:

$$\mathbf{E}\left[\|X - \mathbf{E}[X]\|^2\right] \leq \mathbf{E}\left[\|X\|^2\right]. \tag{38}$$

Proof:

$$\begin{aligned} \mathbf{E}\left[\|X - \mathbf{E}[X]\|^2\right] &= \mathbf{E}\left[\|X\|^2\right] - 2\mathbf{E}\left[\|X\|\right]^2 + \mathbf{E}\left[\|X\|\right]^2 \\ &= \mathbf{E}\left[\|X\|^2\right] - \mathbf{E}\left[\|X\|\right]^2 \leq \mathbf{E}\left[\|X\|^2\right]. \end{aligned} \tag{39}$$

■

APPENDIX B CONVERGENCE PROOF ILLUSTRATED VIA SGD$_\star$

For all VR methods, the first steps of proving convergence are the same. First, we expand

$$\begin{aligned} \|x_{k+1} - x_\star\|^2 &= \|x_k - x_\star - \gamma g_k\|^2 \\ &= \|x_k - x_\star\|^2 - 2\gamma\langle x_k - x_\star, g_k\rangle + \gamma^2\|g_k\|^2. \end{aligned}$$

Now, taking expectation conditioned on x_k and using (6), we arrive at

$$\mathbf{E}_k[\|x_{k+1}-x_\star\|^2] = \|x_k - x_\star\|^2 + \gamma^2\mathbf{E}_k[\|g_k\|^2] - 2\gamma\left\langle x_k - x_\star, \nabla f(x_k)\right\rangle .$$

Using either convexity or strong convexity, we can get rid of the $\langle x_k - x_\star, \nabla f(x_k)\rangle$ term. In particular, since $f(x)$ is μ-strongly convex, we have

$$\mathbf{E}_k[\|x_{k+1}-x_\star\|^2] \leq (1-\mu\gamma)\|x_k - x_\star\|^2 + \gamma^2\mathbf{E}_k[\|g_k\|^2] - 2\gamma(f(x_k)-f(x_\star)). \tag{40}$$

To conclude the proof, we need a bound on the second moment $\mathbf{E}_k[\|g_k\|^2]$ of g_k. For the plain vanilla SGD, often it is simply assumed that this variance term is bounded uniformly by an unknown constant $B > 0$. However, this assumption rarely holds in practice, and even when it does, the resulting convergence speed depends on this unknown constant B. In contrast, for a VR method, we can explicitly control the second moment of g_k since we can control the variance of g_k and

$$\mathbf{E}_k[\|g_k - \nabla f(x_k)\|^2] \;=\; \mathbf{E}\left[\|g_k\|^2\right] - \|\nabla f(x_k)\|^2. \tag{41}$$

To illustrate, we now prove the convergence of SGD$_\star$.

Theorem 1 [22]: Consider the iterates of SGD$_\star$ (13). If Assumptions 1 and 2 hold and $\gamma \leq (1/L_{\max})$, then the iterates converge linearly with

$$\mathbf{E}\left[\|x_{k+1}-x_\star\|^2\right] \;\leq\; (1-\gamma\mu)\mathbf{E}\left[\|x_k - x_\star\|^2\right]. \tag{42}$$

Thus, the iteration complexity of SGD$_\star$ is given by

$$k \geq \frac{L_{\max}}{\mu}\log\left(\frac{1}{\varepsilon}\right) \Rightarrow \frac{\mathbf{E}\left[\|x_k - x_\star\|^2\right]}{\|x_0 - x_\star\|^2} < \varepsilon. \tag{43}$$

Proof: Using Lemma 1, we have

$$\begin{aligned}\mathbf{E}_k[\|g_k\|^2] &= \mathbf{E}_k[\|\nabla f_i(x_k) - \nabla f_i(x_\star)\|^2]\\ &\leq 2L_{\max}(f(x_k) - f(x_\star)).\end{aligned} \tag{44}$$

Using the above in (40), we have

$$\mathbf{E}_k[\|x_{k+1}-x_\star\|^2] \leq (1-\mu\gamma)\|x_k - x_\star\|^2 + 2\gamma(\gamma L_{\max} - 1)(f(x_k) - f(x_\star)). \tag{45}$$

Now, by choosing $\gamma \leq (1/L_{\max})$, we have that $\gamma L_{\max} - 1 < 0$, and consequently, $2\gamma(\gamma L_{\max} - 1)(f(x_k) - f(x_\star))$ is negative since $f(x_k) - f(x_\star) \geq 0$. Thus, it now follows by taking expectation in (45) that

$$\mathbf{E}\left[\|x_{k+1}-x_\star\|^2\right] \leq (1-\mu\gamma)\mathbf{E}\left[\|x_k - x_\star\|^2\right].$$

■

This proof also shows that the shifted SGD method is a VR method. Indeed, since

$$\begin{aligned}\mathbf{E}\left[\|g_k - \nabla f(x_k)\|^2\right] &= \mathbf{E}\left[\|\nabla f_i(x_k) - \nabla f_i(x_\star) - \nabla f(x_k)\|^2\right]\\ &\leq \mathbf{E}\left[\|\nabla f_i(x_k) - \nabla f_i(x_\star)\|^2\right]\\ &\leq 2L_{\max}(f(x_k) - f(x_\star))\end{aligned}$$

where, in the first inequality, we used Lemma 2 with $X = \nabla f_i(x_k) - \nabla f_i(x_\star)$. ■

Acknowledgment

The authors would like to thank Quanquan Gu, Julien Mairal, Tong Zhang, and Lin Xiao for valuable suggestions and comments on an earlier draft of this article. In particular, Quanquan's recommendations for the nonconvex section improved the organization of our Section IV-G.

References

[1] Z. Allen-Zhu, "Katyusha: The first direct acceleration of stochastic gradient methods," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 8194–8244, Jan. 2017.

[2] Z. Allen-Zhu, "Natasha: Faster non-convex stochastic optimization via strongly non-convex parameter," in *Proc. Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 89–97.

[3] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific J. Math.*, vol. 16, no. 1, pp. 1–3, Jan. 1966.

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

[5] C. G. Broyden, "Quasi-Newton methods and their application to function minimisation," *Math. Comput.*, vol. 21, no. 99, pp. 368–381, 1969.

[6] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-Newton method for large-scale optimization," *SIAM J. Optim.*, vol. 26, no. 2, pp. 1008–1031, Jan. 2016.

[7] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, Apr. 2011.

[8] T. Chavdarova, G. Gidel, F. Fleuret, and S. Lacoste-Julien, "Reducing noise in GAN training with variance reduced extragradient," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 393–403.

[9] J. Chen, J. Zhu, Y. W. Teh, and T. Zhang, "Stochastic expectation maximization with variance reduction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7967–7977.

[10] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett, "Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks," *J. Mach. Learn. Res.*, vol. 9, pp. 1775–1822, Jul. 2008.

[11] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. New York, NY, USA: Springer, 2011, pp. 185–212.

[12] D. Csiba, Z. Qu, and P. Richtárik, "Stochastic dual coordinate ascent with adaptive probabilities," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 674–683.

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

[14] A. Defazio and L. Bottou, "On the ineffectiveness of variance reduced optimization for deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1755–1765.

[15] C. Fang, C. J. Li, Z. Lin, and T. Zhang, "SPIDER: Near-optimal non-convex optimization via stochastic path-integrated differential estimator," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 689–699.

[16] R. Fletcher, "A new approach to variable metric algorithms," *Comput. J.*, vol. 13, no. 3, pp. 317–323, 1970.

[17] R. Frostig, R. Ge, S. M. Kakade, and A. Sidford, "Competing with the empirical risk minimizer in a single pass," in *Proc. Conf. Learn. Theory*, 2015, pp. 728–763.

[18] D. Garber and E. Hazan, "Fast and simple PCA via

convex optimization,” 2015, *arXiv:1509.05647*. [Online]. Available: http://arxiv.org/abs/1509.05647

[19] N. Gazagnadou, R. M. Gower, and J. Salmon, “Optimal mini-batch and step sizes for SAGA,” in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, 2019, pp. 2142–2150.

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

[21] P. Gong and J. Ye, “Linear convergence of variance-reduced stochastic gradient without strong convexity,” 2014, *arXiv:1406.1102*. [Online]. Available: http://arxiv.org/abs/1406.1102

[22] E. Gorbunov, F. Hanzely, and P. Richtarik, “A unified theory of SGD: Variance reduction, sampling, quantization and coordinate descent,” in *Proc. Mach. Learn. Res.*, vol. 108, S. Chiappa and R. Calandra, Eds. PMLR, Aug. 2020, pp. 680–690.

[23] R. M. Gower, D. Goldfarb, and P. Richtárik, “Stochastic block BFGS: Squeezing more curvature out of data,” in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 1869–1878.

[24] R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik, “SGD: General analysis and improved rates,” in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, 2019, pp. 5200–5209.

[25] R. M. Gower, P. Richtárik, and F. Bach, “Stochastic quasi-gradient methods: Variance reduction via Jacobian sketching,” *Math. Program.*, May 2020. [Online]. Available: https://link.springer.com/article/10.1007%2Fs10107-020-01506-0

[26] R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen, “Stop wasting my gradients: Practical SVRG,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2251–2259.

[27] T. Hofmann, A. Lucchi, S. Lacoste-Julien, and B. McWilliams, “Variance reduced stochastic gradient descent with neighbors,” in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 2305–2313.

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

[29] H. Karimi, J. Nutini, and M. Schmidt, “Linear convergence of gradient and proximal-gradient methods under the Polyak-Lojasiewicz condition,” in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Springer, 2016, pp. 795–811.

[30] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–13.

[31] J. Konečný, J. Liu, P. Richtárik, and M. Takáč, “Mini-batch semi-stochastic gradient descent in the proximal setting,” *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 2, pp. 242–255, Mar. 2016.

[32] J. Konečný and P. Richtárik, “Semi-stochastic gradient descent methods,” *CoRR*, vols. abs/1312.1666, pp. 1–9, May 2013.

[33] D. Kovalev, S. Horváth, and P. Richtárik, “Don’t jump through hoops and remove those loops: SVRG and Katyusha are better without the outer loop,” in *Proc. 31st Int. Conf. Algorithmic Learn. Theory*, 2020, pp. 451–467.

[34] D. Kovalev, K. Mishchenko, and P. Richtárik, “Stochastic Newton and cubic Newton methods with simple local linear-quadratic rates,” in *Proc. NeurIPS Beyond 1st Order Methods Workshop*, 2019, pp. 1–16.

[35] A. Kulunchakov and J. Mairal, “Estimate sequences for variance-reduced stochastic composite optimization,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3541–3550.

[36] G. Lan and Y. Zhou, “An optimal randomized incremental gradient method,” *Math. Program.*, vol. 171, nos. 1–2, pp. 167–215, Sep. 2018.

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

[38] R. Leblond, F. Pedregosa, and S. Lacoste-Julien, “ASAGA: Asynchronous parallel SAGA,” in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 46–54.

[39] L. Lei and M. Jordan, “Less than a single pass: Stochastically controlled stochastic gradient,” in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 148–156.

[40] X. Lian, M. Wang, and J. Liu, “Finite-sum composition optimization via variance reduced gradient descent,” in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1159–1167.

[41] H. Lin, J. Mairal, and Z. Harchaoui, “Catalyst acceleration for first-order convex optimization: From theory to practice,” *J. Mach. Learn. Res.*, vol. 18, no. 212, pp. 1–54, 2018.

[42] S. Lohr, *Sampling: Design and Analysis*. Duxbury Press, 1999.

[43] M. Mahdavi and R. Jin, “MixedGrad: An O(1/T) convergence rate algorithm for stochastic smooth optimization,” 2013, *arXiv:1307.7192*. [Online]. Available: http://arxiv.org/abs/1307.7192

[44] J. Mairal, “Incremental majorization-minimization optimization with application to large-scale machine learning,” *SIAM J. Optim.*, vol. 25, no. 2, pp. 829–855, Jan. 2015.

[45] Y. Malitsky and K. Mishchenko, “Adaptive gradient descent without descent,” 2019, *arXiv:1910.09529*. [Online]. Available: http://arxiv.org/abs/1910.09529

[46] A. Mokhtari and A. Ribeiro, “RES: Regularized stochastic BFGS algorithm,” *IEEE Trans. Signal Process.*, vol. 62, no. 23, pp. 1109–1112, Dec. 2014.

[47] A. Mokhtari and A. Ribeiro, “Global convergence of online limited memory BFGS,” *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 3151–3181, 2015.

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

[49] D. Needell, N. Srebro, and R. Ward, “Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm,” *Math. Program.*, vol. 155, nos. 1–2, pp. 549–573, Jan. 2016.

[50] Y. Nesterov, “A method for solving a convex programming problem with convergence rate $O(1/k^2)$,” *Sov. Math. Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[51] Y. Nesterov, “Efficiency of coordinate descent methods on huge-scale optimization problems,” *SIAM J. Optim.*, vol. 22, no. 2, pp. 341–362, Jan. 2012.

[52] Y. Nesterov, *Introductory Lectures Convex Optimization: A Basic Course*, 2nd ed. New York, NY, USA: Springer, 2014.

[53] Y. Nesterov and T. B. Polyak, “Cubic regularization of newton method and its global performance,” *Math. Program.*, vol. 108, pp. 177–205, Apr. 2006.

[54] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, “SARAH: A novel method for machine learning problems using stochastic recursive gradient,” in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2613–2621.

[55] B. Palaniappan and F. Bach, “Stochastic variance reduction methods for saddle-point problems,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1416–1424.

[56] M. Papini, D. Binaghi, G. Canonaco, M. Pirotta, and M. Restelli, “Stochastic variance-reduced policy gradient,” in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 4026–4035.

[57] B. Polyak, “Gradient methods for the minimisation of functionals,” in *Proc. USSR Comput. Math. Math. Phys.*, vol. 3, 1963, pp. 864–878.

[58] X. Qian, Z. Qu, and P. Richtárik, “SAGA with arbitrary sampling,” in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 5190–5199.

[59] Z. Qu, P. Richtárik, M. Takáč, and O. Fercoq, “SDNA: Stochastic dual Newton ascent for empirical risk minimization,” in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1823–1832.

[60] Z. Qu, P. Richtárik, and T. Zhang, “Quartz: Randomized dual coordinate ascent with arbitrary sampling,” in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 865–873.

[61] S. J. Reddi, A. Hefny, S. Sra, B. Póczos, and A. Smola, “Stochastic variance reduction for nonconvex optimization,” in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 314–323.

[62] S. J. Reddi, S. Sra, B. Poczos, and A. Smola, “Fast incremental method for smooth nonconvex optimization,” in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, Dec. 2016, pp. 1971–1977.

[63] P. Richtárik and M. Takáč, “Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function,” *Math. Program.*, vol. 144, no. 1, pp. 1–38, Dec. 2012.

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

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

[66] M. W. Schmidt, R. Babanezhad, M. O. Ahmed, A. Defazio, A. Clifton, and A. Sarkar, “Non-uniform stochastic average gradient method for training conditional random fields,” in *Proc. 18th Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2015, pp. 819–828.

[67] N. N. Schraudolph and G. Simon, “A stochastic quasi-Newton method for online convex optimization,” in *Proc. 11th Int. Conf. Artif. Intell. Statist.*, 2007, pp. 436–443.

[68] O. Sebbouh, N. Gazagnadou, S. Jelassi, F. Bach, and R. M. Gower, “Towards closing the gap between the theory and practice of SVRG,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 646–656.

[69] S. Shalev-Shwartz, “SDCA without duality, regularization, and individual convexity,” in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 747–754.

[70] S. Shalev-Shwartz and T. Zhang, “Stochastic dual coordinate ascent methods for regularized loss,” *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 567–599, 2013.

[71] S. Shalev-Shwartz and T. Zhang, “Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization,” in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 64–72.

[72] O. Shamir, “A stochastic PCA and SVD algorithm with an exponential convergence rate,” in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, vol. 37, 2015, pp. 144–152.

[73] D. F. Shanno, “Conditioning of quasi-Newton methods for function minimization,” *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1971.

[74] T. Strohmer and R. Vershynin, “A randomized Kaczmarz algorithm with exponential convergence,” *J. Fourier Anal. Appl.*, vol. 15, no. 2, p. 262, 2009.

[75] M. Takáč, A. Bijral, P. Richtárik, and N. Srebro, “Mini-batch primal and dual methods for SVMs,” in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 537–552.

[76] D. Vainsencher, H. Liu, and T. Zhang, “Local smoothness in variance reduced optimization,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2179–2187.

[77] H.-T. Wai, M. Hong, Z. Yang, Z. Wang, and K. Tang, “Variance reduced policy evaluation with smooth function approximation,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5776–5787.

[78] X. Wang, S. Ma, D. Goldfarb, and W. Liu, “Stochastic quasi-Newton methods for nonconvex stochastic optimization,” *SIAM J. Optim.*, vol. 27, no. 2, pp. 927–956, 2017.

[79] Z. Wang, Y. Zhou, Y. Liang, and G. Lan, “Stochastic variance-reduced cubic regularization for nonconvex optimization,” 2018, *arXiv:1802.07372*. [Online]. Available: http://arxiv.org/abs/1802.07372

[80] B. E. Woodworth and N. Srebro, “Tight complexity bounds for optimizing composite objectives,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3639–3647.

[81] L. Xiao and T. Zhang, “A proximal stochastic gradient method with progressive variance reduction,” *SIAM J. Optim.*, vol. 24, no. 4, pp. 2057–2075, Jan. 2014.

[82] P. Xu, F. Gao, and Q. Gu, “An improved convergence analysis of stochastic variance-reduced policy gradient,” in *Proc. 35th Conf. Uncertainty Artif. Intell.*, 2019, pp. 541–551.

[83] J. Zhang and L. Xiao, "Stochastic variance-reduced prox-linear algorithms for nonconvex composite optimization," Microsoft, Albuquerque, NM, USA, Tech. Rep. MSR-TR-2020-11, 2020.

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

[85] Y. Zhang and L. Xiao, "Stochastic primal-dual coordinate method for regularized empirical risk minimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2939–2980, Jan. 2017.

[86] L. W. Zhong and J. T. Kwok, "Fast stochastic alternating direction method of multipliers," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, 2014, pp. 46–54.

[87] D. Zhou and Q. Gu, "Lower bounds for smooth nonconvex finite-sum optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7574–7583.

[88] D. Zhou, P. Xu, and Q. Gu, "Stochastic nested variance reduced gradient descent for nonconvex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3921–3932.

[89] D. Zhou, P. Xu, and Q. Gu, "Stochastic variance-reduced cubic regularization methods," *J. Mach. Learn. Res.*, vol. 20, no. 134, pp. 1–47, 2019.

[90] D. Zou, P. Xu, and Q. Gu, "Stochastic gradient Hamiltonian Monte Carlo methods with recursive variance reduction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 3835–3846.

ABOUT THE AUTHORS

Robert M. Gower received the bachelor's and master's degrees in applied mathematics from the State University of Campinas, Campinas, Brazil, in 2009 and 2011, respectively.

He is currently a Visiting Researcher with Facebook AI Research, New York, NY, USA. He joined Télécom Paris, Paris, France, as an Assistant Professor, in 2017. He is interested in designing and analyzing new stochastic algorithms for solving big data problems in machine learning and scientific computing. He designed the current state-of-the-art algorithms for automatically calculating high-order derivatives using backpropagation.

Mark Schmidt is currently an Associate Professor with the Department of Computer Science, The University of British Columbia, Vancouver, BC, Canada. His research focuses on machine learning and numerical optimization.

Dr. Schmidt is the Canada Research Chair, the Alfred P. Sloan Fellow, and the CIFAR Canada AI Chair with the Alberta Machine Intelligence Institute (Amii) and was awarded the SIAM/MOS Lagrange Prize in continuous optimization with Nicolas Le Roux and Francis Bach.

Francis Bach graduated from École Polytechnique, Palaiseau, France, in 1997. He received the Ph.D. degree in computer science from the University of California at Berkeley, Berkeley, CA, USA, in 2005, working with Prof. Michael Jordan.

He is currently a Researcher with Inria, Paris, France, leading the Machine Learning Team, since 2011, which is a part of the Computer Science Department, École Normale Supérieure. He spent two years in the Mathematical Morphology Group, École des Mines de Paris, Paris. He was with the Computer Vision Project-Team, Inria/École Normale Supérieure, from 2007 to 2010. He is primarily interested in machine learning, and especially in sparse methods, kernel-based learning, large-scale optimization, computer vision, and signal processing.

Dr. Bach obtained a Starting Grant in 2009 and a Consolidator Grant in 2016 from the European Research Council. He received the Inria Young Researcher Prize in 2012, the ICML Test-of-Time Award in 2014, the Lagrange Prize in continuous optimization in 2018, and the Jean-Jacques Moreau Prize in 2019. In 2015, he was the Program Co-Chair of the International Conference in Machine learning (ICML) and the General Chair in 2018. He is also the Co-Editor-in-Chief of the *Journal of Machine Learning Research*.

Peter Richtárik received the Ph.D. degree from Cornell University, Ithaca, NY, USA, in 2007.

He was a Postdoctoral Fellow with the Université Catholique de Leuven, Leuven, Belgium, before joining The University of Edinburgh, Edinburgh, U.K., in 2009, and the King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia, in 2017. He is currently a Professor of computer science and mathematics with KAUST. He is also an EPSRC Fellow in mathematical sciences, a Fellow of the Alan Turing Institute, and is with the Visual Computing Center and the Extreme Computing Research Center, KAUST. His research interests lie at the intersection of mathematics, computer science, machine learning, optimization, numerical linear algebra, and high-performance computing. Through his recent work on randomized decomposition algorithms (such as randomized coordinate descent methods, stochastic gradient descent methods and their numerous extensions, improvements, and variants), he has contributed to the foundations of the emerging field of big data optimization, randomized numerical linear algebra, and stochastic methods for empirical risk minimization.

Dr. Richtárik's articles attracted international awards to his collaborators, including the SIAM SIGEST Best Paper Award, the IMA Leslie Fox Prize (second prize, three times), and the INFORMS Computing Society Best Student Paper Award (sole runner up).