# Variance-Reduced Methods for Machine Learning

Robert M Gower, Mark Schmidt, Francis Bach, Peter Richtárik

# Variance-Reduced Methods for Machine Learning

**Robert M. Gower**[a], **Mark Schmidt**[b], **Francis Bach**[c], **and Peter Richtárik**[d]

[a]LTCI, Télécom Paris, Institut Polytechnique de Paris ; [c]Inria - PSL Research University, France; [b]University of British Columbia, CCAI Affiliate Chair (Amii), Canada; [d]King Abdullah University of Science and Technology, Kingdom of Saudi Arabia

**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 8 years have seen an exciting new development: variance reduction (VR) for stochastic optimization methods. These VR methods excel in settings where more than one pass through the training data is allowed, achieving a faster convergence than SGD in theory as well as 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 non-expert readers. We focus mainly on the convex setting, and leave pointers to readers interested in extensions for minimizing non-convex functions.**

optimization, machine learning, variance reduction

## 1. Introduction

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

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

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 $i$-th data point,

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

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 fits perfectly the $i$-th data point. The function $f(x)$ represents the average *loss* of our model over the full dataset. A problem of the form [2] characterizes the training of not only linear least squares, but many models studied in machine learning. For example, the logistic regression model solves

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

where we are now considering a binary classification task with $b_i \in \{-1, +1\}$ (and predictions are made using the sign of $a_i^\top x$). Here, we have also used $\frac{\lambda}{2}\|x\|^2 := \frac{\lambda}{2}\sum_{i=1}^{d} x_i^2$, 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 $f_i(x) + \frac{\lambda}{2}\|x\|^2$. The training procedure in most supervised machine learning models can be written in the form [2], including L1-regularized least squares, support vector machines, 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 datasets 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 datasets is to use *stochastic gradient descent* (SGD) methods, which use a few randomly chosen data points in each of their iterations. Further, there has been a recent surge in interest in variance-reduced (VR) stochastic gradient methods which converge faster than classic stochastic gradient methods.

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

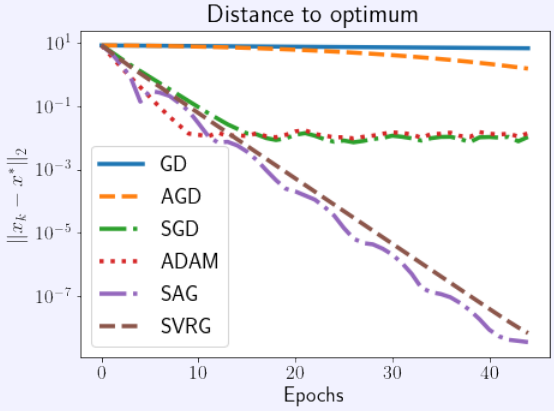



**Fig. 1.** Comparison of the GD, AGD (Accelerated GD (Nesterov, 1983)), SGD and ADAM (Kingma and Ba, 2015) methods to the VR methods SAG and SVRG on a logistic regression problem based on the mushrooms data set (Chang and Lin, 2011), where $n = 8,124$ and $d = 112$.

**A. Gradient and Stochastic Gradient Descent.** The classic GD (gradient descent) method applied to problem [2] takes the form

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

where $\gamma > 0$ is a fixed stepsize*. At each iteration the GD method needs to calculate a gradient $\nabla f_i(x_k)$ for every $i$th

*The classic way to implement GD is to determine $\gamma$ 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 (Armijo, 1966; Malitsky and Mishchenko, 2019). This line search requires multiple

[2]To whom correspondence should be addressed. E-mail: gower.robert@gmail.com

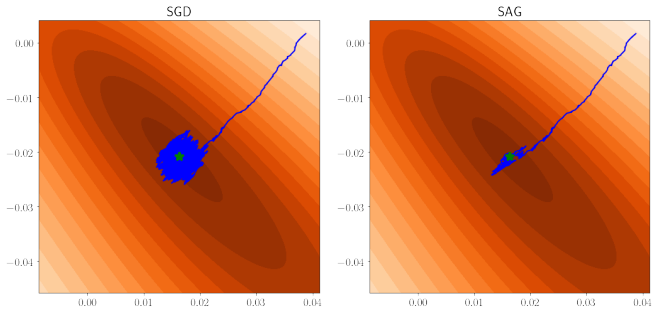



**Fig. 2.** Level set plot of 2D logistic regression with the iterates of SGD (left) and SAG (right) with constant stepsize. The green star is the $x_*$ solution.

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 *stochastic gradient descent* (SGD) method,

$$x_{k+1} = x_k - \gamma \nabla f_{i_k}(x_k), \qquad [5]$$

first introduced by Robbins and Monro (1951). 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 Figure 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 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. The Issue with Variance.** Observe that if we choose the random index $i_k \in \{1, \dots, n\}$ 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

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

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 Figure 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 (Chang and Lin, 2011). The concentric ellipses in Figure 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 SAG (that we present later) in Figure 2 using the same constant stepsize.

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 multiples times. This is why we use fixed constant stepsize instead.

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 $x_k \to x_\star$.

**C. Classic Variance Reduction Methods.** There are several classic techniques for dealing with the non-convergence due to the variance in the $\nabla f_i(x_k)$ values. For example, Robbins and Monro (1951) address the issue of the variance using a sequence of decreasing stepsizes $\gamma_k$. 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 to early (before reaching the solution) or to 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 *mini-batching*, and is especially useful when multiple gradients can be evaluated in parallel. This leads to an iteration of the form

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

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.[†] 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

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

$$x_{k+1} = x_k - \gamma m_k, \qquad [9]$$

where the momentum parameter $\beta$ 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,

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

Thus $m_k$ is a weighted sum of the stochastic gradients. Moreover since $\sum_{t=0}^{k} \beta^{k-t} = \frac{1-\beta^{k+1}}{1-\beta}$, we have that $\frac{1-\beta}{1-\beta^{k-1}} m_k$ is a weighted average of stochastic gradients. If we compare this with the expression of the full gradient which is a plain average, $\nabla f(x_k) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_k)$, we can interpret $\frac{1-\beta}{1-\beta^k} m_k$ (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 variance reduced method we will see in Section 2.A contours this issue by using a plain average, as opposed to any weighted average.

[†] If we sample without replacement the variance decreases at a faster rate (see Section 2.7 in Lohr (1999)), and with $|B_k| = n$ the variance is zero.

**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 $\nabla f(x_k)$, variance-reduced 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

$$x_{k+1} = x_k - \gamma g_k, \qquad [11]$$

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[‡]

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

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 variance-reduced 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 Figure 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

$$x_{k+1} = x_k - \gamma\left(\nabla f_{i_k}(x_k) - \nabla f_{i_k}(x_\star)\right), \qquad [13]$$

called SGD$_\star$ (**?**). 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. Further, 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 $\nabla f(x_\star) = 0$,

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

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

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

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

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

where we used Lemma A.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.

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

**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 1.1.** *The function $f$ is differentiable and $L$-smooth, meaning that*

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\| \qquad [14]$$

*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 $L_{\max} := \max\{L_1, \dots, L_n\}$.*

While this is typically viewed as a weak assumption, in Section 4 we comment on VR methods that apply to non-smooth 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 $L$, $|f''(x)| \le L$ 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 1.2.** *The function $f$ is $\mu$-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 $i = 1, \dots, n$.*

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 $A := [a_1, \dots, a_n]$ has full row rank. On other hand, the L2-regularized logistic regression problem [3] satisfies this assumption with $\mu \ge \lambda$ due to the presence of the regularizer. As we detail in Section 4, it is possible to relax the strong convexity assumption as well as the assumption that each $f_i$ is convex.

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

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

in the case when each "loss" function $\ell_i : \mathbb{R} \mapsto \mathbb{R}$ is twice-differentiable with $\ell_i''$ 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, Huber robust regression, and a variety of others. In this setting, for all $i$ we have $L_i \le M\|a_i\|^2 + \lambda$ and $\mu \ge \lambda$.

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 or equal to one, and when it is significantly larger than one, the level sets of the function become very elliptical which causes the iterates

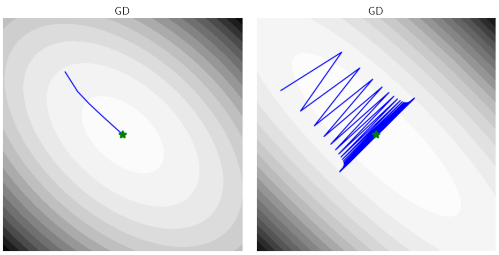



**Fig. 3.** Here we graph the level sets two logistic regression loss functions. The left level sets are each of a well-conditioned logistic function with $\kappa \approx 1$. The right level sets are of an ill-conditioned logistic functions with $\kappa \gg 1$.

of the GD method to oscillate. This is illustrated in Figure 3. In contrast, when $\kappa$ is close to 1, GD converges quickly.

Under Assumptions 1.1 and 1.2, 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

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

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

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

The smallest $k$ satisfying this inequality is known as the *iteration complexity* of the algorithm. Below 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.[§] But the VR methods are superior in terms of total runtime since each iteration of gradient descent costs $n$-times more than an iteration of a VR method[¶]. Classic SGD methods have the advantage that their runtime and their convergence rate does not depend on $n$, but it does have a much worse dependency on the tolerance $\varepsilon$ which explains SGD's poor performance when the tolerance is small. [‖]

[§] In Section 4 we discuss how non-uniform sampling within VR methods leads to a faster rate, depending on the mean $\bar{L} := \frac{1}{n}\sum_i L_i$ rather than on the maximum $\max_i L_i$.

[¶] And since $L_{\max} \leq nL$.

[‖] We have omitted an additional term for the SGD iteration complexity of the form $\mathcal{O}(\sigma^2/\mu\varepsilon)$, where $\sigma^2 \geq \mathbf{E}_i\left[\|\nabla f_i(x_*)\|^2\right]$, see Theorem 2.1 in Needell et al. (2015)

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

## 2. Basic Variance-Reduced Methods

The first wave of variance-reduced methods that achieve the convergence rate from the previous section started with the stochastic average gradient (SAG) method (Le Roux et al., 2012; Schmidt et al., 2017). This was followed shortly after by the stochastic dual coordinate ascent (SDCA) (Richtárik and Takáč, 2014; Shalev-Shwartz and Zhang, 2013), MISO (Mairal, 2015), stochastic variance-reduced gradient (SVRG/S2GD) (Mahdavi and Jin, 2013; Johnson and Zhang, 2013; Konecný and Richtárik, 2013; Zhang et al., 2013), and SAGA (stochastic average gradient "amélioré") (Defazio et al., 2014) methods. In this section we present several of these original methods, while Section 4 covers more recent methods that offer improved properties in certain settings over these original methods.

**A. Stochastic Average Gradient (SAG).** 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 variance-reduced method: the *stochastic average gradient* (SAG) method.

The stochastic average gradient (SAG) method (Le Roux et al., 2012; Schmidt et al., 2017) is a stochastic variant of the earlier incremental aggregated gradient (IAG) method (Blatt et al., 2007). 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

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

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

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

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 we sample the index $i_k$ on iteration $k$. It follows from [18] and [19] that

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

Since the $v_{k-1}^j$ 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.

**Algorithm 1** SAG: Stochastic Average Gradient method

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

The SAG method was the first stochastic methods 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 Figure 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 obtain 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 Schmidt et al. (2017) 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 $i = 1, \dots, n$. Using these vectors we can rewrite our full gradient as

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

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 $i \in \{1, \dots, n\}$. That is, we can solve [2] by applying the SGD method with the gradient estimate

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

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 $\mathbf{E}_{i\sim\frac{1}{n}}\left[v^i\right] = \frac{1}{n}\sum_{j=1}^n v^j$ in [12] gives

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

where we used Lemma A.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 $\text{SGD}_\star$ method [13] also implements this approach with $v^i = \nabla f_i(x_\star)$. But 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

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

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 $j \in \{1, \dots, n\}$, 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 (Algorithm 1) except now we store the previously known gradient of $f_{i_k}$ in a dummy variable $v^{\text{old}}$ 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 $i = 1, \dots, n$
**for** $k = 1, \dots, T-1$ **do**
  Sample $i_k \in \{1, \dots, n\}$
  $v^{\text{old}} = v^{i_k}$
  $v^{i_k} = \nabla f_{i_k}(x_k)$
  $x_{k+1} = x_k - \gamma\left(v^{i_k} - v^{\text{old}} + \bar{g}_k\right)$
  $\bar{g}_k = \bar{g}_{k-1} + \frac{1}{n} v^{i_k} - \frac{1}{n} v^{\text{old}}$
**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 $i = 1, \dots, n$ which amounts to an $\mathcal{O}(nd)$ storage. This can be infeasible when both $d$ and $n$ are large. We detail in Section 3 how we can reduce this memory requirement for common models like 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 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 (Johnson and Zhang, 2013; Mahdavi and Jin, 2013; Zhang et al., 2013). 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 stochastic variance-reduced gradient (SVRG) method, following the naming of (Johnson and Zhang, 2013; Zhang et al., 2013).

We formalize the SVRG method in Algorithm 3. Using [23] we have that the variance of the gradient estimate $g_k$ is bounded by

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

where the second inequality uses the $L_i$-smoothness of each $L_i$[**]. 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 3 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.

**Algorithm 3** SVRG: Stochastic Variance-Reduced Gradient 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: $\bar{x}_s = x_t$.

Johnson and Zhang (2013) 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 dependency must hold between $L_{\max}$, $\mu$, the stepsize $\gamma$, 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 Figure 1.

There are now many variations on the original SVRG method. For example, there are variants that use alternative distributions for $t$ (Konecný and Richtárik, 2013) and variants that allow stepsizes of the form $\mathcal{O}(1/L_{\max})$ (Hofmann et al., 2015; Kulunchakov and Mairal, 2019; Kovalev et al., 2020). There are also variants that use a mini-batch approximation of $\nabla f(\bar{x})$ to reduce the cost of these full-gradient evaluations, and that grow the mini-batch size in order to maintain the VR property (Harikandeh et al., 2015; Frostig et al., 2015). And there are variants that repeatedly update $g_k$ in the inner loop according to (Nguyen et al., 2017)

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

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 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 stochastic dual coordinate ascent (SDCA) method (Shalev-Shwartz and Zhang, 2013) which extended recent work on coordinate descent methods to the finite-sum problem.[††]

The intuition behind SDCA, and its variants, is that the coordinates of the gradient provide a naturally variance-reduced estimate of the gradient. That is, let $j \in \{1, \ldots, d\}$ and let $\nabla_j f(x) := \frac{\partial f(x)}{\partial x_j} e_j$ be the *coordinate-wise* derivative of $f(x)$, where $e_j \in \mathbb{R}^d$ is the $j$th unit coordinate vector. An important feature of coordinate-wise 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 $\nabla f_j$ that may be different than zero at $x_\star$. Due to this feature, we have that

$$\|\nabla f(x) - \nabla_j f(x)\|^2 \underset{x \to x_\star}{\longrightarrow} 0, \qquad [26]$$

and thus coordinate-wise 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 $\{1, \ldots, d\}$. Thus for any given $i \in \{1, \ldots, d\}$ we have that $\mathbf{P}[j = i] = \frac{1}{d}$. Consequently $d \times \nabla_j f(x)$ is an unbiased estimate of $\nabla f(x)$ since

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

Thus $\nabla_j f(x)$ has all the favorable properties we would like for a variance reduced estimate of the full gradient without using covariates. The downside of using this coordinate-wise

[**] When each $f_i$ is also convex, we can derive the bound

$$\mathbb{E}[\|\nabla f(\bar{x}_{s-1}) - \nabla f(x_k)\|^2] \leq 4L_{\max}(f(x_k) - f(x^*) + f(\bar{x}_{s-1}) - f(x^*))$$

using analogous proof to Lemma A.1. This bound on the variance is key to proving a good convergence rate for SVRG in the convex setting.

[††] The modern interest in coordinate ascent/descent methods began with Nesterov (2012), which considered coordinate-wise gradient descent 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 coordinate-wise gradient descent steps (Richtárik and Takáč, 2014)

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

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

So it would seem that using coordinate-wise 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 coordinate-wise derivatives can make use of the inherit structure.

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

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

where $\ell_i^\star(v) := \sup_x\{\langle x, v\rangle - f(x)\}$ is the *convex conjugate* of $\ell_i$. 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.[‡‡] Indeed, Shalev-Shwartz and Zhang (2013) 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`.[§§] 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 $n$. Further, we can use a one-dimensional 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 (Shalev-Shwartz, 2016). 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 the next section). Variants of `SDCA` that apply 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 $\mu$ is equal to $\lambda$. For problems where $\mu$ is greater than $\lambda$, the primal `VR` methods often significantly outperform `SDCA`.

## 3. Practical Considerations

In order to implement the basic `VR` methods and 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 stepsize 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-gradient descent methods is the *Armijo line-search* (Armijo, 1966). Given a current point $x_k$ and a search direction $g_k$, the Armijo line-search for a $\gamma_k$ that is on the line $\gamma_k \in \{\gamma \ : \ x_k + \gamma g_k\}$ and such that gives a sufficient decrease of the function

$$f(x_k + \gamma_k g_k) < f(x_k) - c\gamma_k \|\nabla f(x_k)\|^2. \quad [28]$$

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

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

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

This is used in the implementation of Schmidt et al. (2017) with $c = \frac{1}{2}$ 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 (2015) considers the "Bottou trick" for setting the stepsize in practice.[¶¶] This method takes a small sample of the dataset (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 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-gradient descent 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`

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

[§§] This iteration complexity is in terms of the duality gap. Related results for certain problem structures include Strohmer and Vershynin (2009); Collins et al. (2008).

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

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 (Mairal, 2015), based on a quadratic lower bound (Lin et al., 2018).

**C. Reducing Memory Requirement.** Although SVRG removes the memory requirement of earlier VR methods, in practice SAG/SAGA require 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 (Schmidt et al., 2015).

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

The first one, described by Schmidt et al. (2017, Section 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 non-zeroes. 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 non-zero 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. (2017, Section 2) for SAGA, adds an extra randomness to the update $x_{k+1} = x_k - \gamma\left(\nabla f_{i_k}(x_k) - \nabla f_{i_k}(\bar{x}_{i_k}) + \bar{g}_k\right)$, 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. (2017).

## 4. 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 $\rho$ of the VR methods depends on the the number of training examples $n$. This is in contrast to the convergence rates of classic SGD methods, which 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 Figure 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 $n$. Le Roux et al. (2012) and Konecný and Richtárik (2013) 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[***].

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 (2017) show that this "best of both worlds" result can be achieved for the "practical" SVRG variant where we use a growing mini-batch approximation of $\nabla f(\bar{x})$.

**B. Non-Uniform 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 non-uniform sampling of the random training example $i_k$. In particular, these algorithms bias the choice of $i_k$ towards 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 $\kappa_{\text{mean}} := (\frac{1}{n}\sum_i L_i)/\mu$ rather than on $\kappa_{\max} = (\max_i L_i)/\mu$. This improved rate under non-uniform sampling has been shown for the basic VR methods SVRG (Xiao and Zhang, 2014), SDCA (Qu et al., 2015), and SAGA (Schmidt et al., 2015; Gower et al., 2018).

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

[***] The implementation of Schmidt et al. (2017) does not use this trick. Instead, it replaces $n$ in Line 7 of Algorithm 1 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.

by (Csiba et al., 2015) and is based on updating the probabilities in SDCA by using what is known as the *dual residue*.

Schmidt et al. (2017) 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. (2015).

**C. Mini-batching.** Another strategy to improve the dependence on the $L_i$ values is to use mini-batching, analogous to the classic mini-batch SGD method [7], to obtain a better approximation of the gradient. There are a number of ways of doing mini-batching, 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 $B_k \subset \{1, \dots, n\}$ 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 mini-batch estimate given by $\frac{1}{b}\sum_{i \in B_k} \nabla f_i(x^k)$.

There were a variety of early mini-batch methods (Takáč et al., 2013; Konečný et al., 2016; Harikandeh et al., 2015; Hofmann et al., 2015), but the most recent methods are able to obtain an iteration complexity of the form (Qu et al., 2015; Qian et al., 2019; Gazagnadou et al., 2019; Sebbouh et al., 2019)

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

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

$$\mathcal{L}(b) \;=\; \frac{1}{b}\frac{n-b}{n-1}L_{\max} + \frac{n}{b}\frac{b-1}{n-1}L, \qquad [31]$$

is a *mini batch* smoothness constant first defined in (Gower et al., 2018, 2019). This iteration complexity interpolate between the complexity of full-gradient descent when $\mathcal{L}(n) = L$ and VR methods where $\mathcal{L}(1) = L_{\max}$. Since $L \leq L_{\max} \leq nL$ it is possible that $L \ll L_{\max}$. Thus using larger mini batches 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 $L \leq M\lambda_{\max}(\mathbf{A}\mathbf{A}^\top)$ where $\mathbf{A} = [a_1, \dots, a_n] \in \mathbb{R}^{d\times n}$ and $\lambda_{\max}(\cdot)$ is the largest eigenvalue function. The largest eigenvalue can be computed using a reduced SVD 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 mini batches. In such a case, we could replace $L$ with the upper bound $\frac{1}{n}\sum_{i=1}^n L_i$. But 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* (aka *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 $\mathcal{O}(\sqrt{\kappa}\log(1/\varepsilon))$ (Nesterov, 1983). 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 we can hope to achieve is $\mathcal{O}((\sqrt{n\kappa_{\max}} + n)\log(1/\varepsilon))$ (Woodworth and Srebro, 2016; Lan and Zhou, 2018), which was first achieved by an accelerated SDCA method (Shalev-Shwartz and Zhang, 2014). 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, e.g., Shalev-Shwartz and Zhang, 2014; Zhang and Xiao, 2017; Allen-Zhu, 2017a). Moreover, the "catalyst" framework of Lin et al. (2018) 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 non-smooth. This is because the dual remains a smooth problem. A classic example is the support vector machine (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 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 (Chang and Lin, 2011) uses a dual coordinate ascent method.

One of the first ways to handle non-smooth 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 $\Omega$ is convex on its domain. But the function $\Omega$ may be non-smooth and may enforce constraints on $x$. However, $\Omega$ must be "simple" in the sense that it is possible to efficiently compute its proximal operator applied to step of GD, that is

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

should be relatively inexpensive to compute. The above method is known as the proximal-gradient algorithm (see, e.g., Combettes and Pesquet, 2011), and it achieves the iteration complexity $\mathcal{O}(\kappa\log(1/\varepsilon))$ even though $\Omega$ (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 (Xiao and Zhang, 2014; Defazio et al., 2014).

Several authors have also explored combinations of VR methods with the alternating direction method of multipliers (ADMM) approaches (Zhong and Kwok, 2014), which can achieve improved rates in some cases where $\Omega$ does not admit an efficient proximal operator. Several authors have also considered the case where the individual $f_i$ may be non-smooth, replacing them with a smooth approximation (see Allen-Zhu, 2017a). This smoothing approach does not lead to linear convergence rates, but leads to faster sublinear rates than are obtained with SGD methods on non-smooth 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 $\mathcal{O}(1/k)$ (Schmidt et al., 2017; Mairal, 2015; Konecný and Richtárik, 2013; Mahdavi and Jin, 2013; Defazio et al., 2014). 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 like the PL-inequality and KL-inequality (Polyak, 1963), that include standard problems like (unregularized) least squares but where it is still possible to show linear convergence (Gong and Ye, 2014; Karimi et al., 2016; Reddi et al., 2016b,a).

**G. Non-convex Problems.** Since 2014 a sequence of papers 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 non-convex 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 Zhou and Gu (2019) and Fang et al. (2018).

The first assumption 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 non-convex (Shamir, 2015; Garber and Hazan, 2015). Both Garber and Hazan (2015) and Shalev-Shwartz (2016) then showed how to use the catalyst framework (Lin et al., 2018) 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 non-convex $f_i$ functions so long as their average $f$ is $\mu$–strongly convex. Recently, this complexity was shown to match the lower bound in this setting (Zhou and Gu, 2019). Allen-Zhu (2017b) 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 non-convexity" (strong convexity but with a negative parameter $-\mu$). To tackle this setting, Allen-Zhu (2017b) proposed an accelerated variant of SVRG that achieves state-of-the-art complexity results which recently have been shown to be optimal (Zhou and Gu, 2019). [†††]

Even more recently, the convexity assumptions on $f$ have been completely dropped. By only assuming that the $f_i$'s are smooth and $f$ has a lower bound, Fang et al. (2018) 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 $\mathcal{O}\left(n + \sqrt{n}/\varepsilon^2\right)$ iterations. Concurrently to this, Zhou et al. (2018) presented a more involved variant of SVRG that uses multiple reference points and achieves the same iteration complexity. Fang et al. (2018) also provided a lower bound showing that the preceding complexity is optimal under these assumptions.

An interesting source of non-convex functions are problems where the objective is a composition of functions $f(g(x))$, where $g : \mathbb{R}^d \to \mathbb{R}^m$ is a mapping. Even when both $f$ and $g$ are convex, their composition may be non-convex. 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 (Lian et al., 2017). In this setting, the finite sum structures can also be exploited to develop variance reduced 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 (Zhang and Xiao, 2020).

**H. Second-Order Variants.** Inspired by Newton's method, there are now second-order variants of the VR methods of the form

$$x_{k+1} = x_k - \gamma_k H_k g_k, \qquad [33]$$

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, for 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. 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 (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1971).

The first stochastic BFGS method that makes use of subsampling was the online L-BFGS method (Schraudolph and Simon, 2007) which uses subsampled gradients to approximate Hessian-vector products. The regularized BFGS method (Mokhtari and Ribeiro, 2014, 2015) 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 (Byrd et al., 2016). Moritz et al. (2016) then proposed combining SQN with SVRG. The resulting method performs very well in numerical tests and was the first example of a 2nd 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. (2016)'s method was later extended to a block quasi-Newton variant and analysed with an improved complexity by Gower et al. (2016). There are also specialized variants for the non-convex setting (Wang et al., 2017). These 2nd order quasi-Newton variants of the VR methods are hard to analyse and, as far as we are aware, there exists no quasi-Newton variants of [33] that have an update cost independent of $n$ and is proven to have a better global complexity than the $\mathcal{O}((\kappa_{\max} + n)\log(1/\varepsilon))$ of VR methods.

However, there do exist stochastic Newton type methods, such as *Stochastic Dual Newton Ascent* (SDNA) (Qu et al., 2016) that perform 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. (2019) preposed 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 mini-batch size.

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

Yet another line of stochastic second order methods is being developed by combining variance reduction techniques with the cubic regularized Newton method (Nesterov and Polyak, 2006) . These methods replace both the Hessian and gradient by variance reduced 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 a second-order stationary points for smooth non-convex problems Zhou et al. (2019); Wang et al. (2018). This is currently a very active direction of research.

## 5. Discussion and Limitations

In the previous section, 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 mini-batching and acceleration while using a proximal-gradient framework to address non-smooth 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, then doing 1 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 to SGD (Frostig et al., 2015; Harikandeh et al., 2015).

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 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 (Defazio and Bottou, 2019). On the other hand, VR methods are now finding applications in a variety of other machine learning applications including policy evaluation for reinforcement learning (Wai et al., 2019; Papini et al., 2018; Xu et al., 2019), expectation maximization (Chen et al., 2018), simulations using Monte Carlo methods (Zou et al., 2019), saddle-point problems (Palaniappan and Bach, 2016), and generative adversarial networks (Chavdarova et al., 2019).


## Acknowledgement

We thank Quanquan Gu, Julien Mairal, Tong Zhang and Lin Xiao for valuable suggestions and comments on an earlier draft of this paper. In particular, Quanquan's recommendations for the non-convex section improved the organization of our Section G.


## References


Allen-Zhu, Z. (2017a). Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244.

Allen-Zhu, Z. (2017b). Natasha: Faster non-convex stochastic optimization via strongly non-convex parameter. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 89–97.

Armijo, L. (1966). Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3.

Blatt, D., Hero, A. O., and Gauchman, H. (2007). A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51.

Broyden, C. G. (1967). Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381.

Byrd, R. H., Hansen, S. L., Nocedal, J., and Singer, Y. (2016). A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031.

Chang, C. C. and Lin, C. J. (2011). LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27.

Chavdarova, T., Gidel, G., Fleuret, F., and Lacoste-Julien, S. (2019). Reducing noise in gan training with variance reduced extragradient. In *Advances in Neural Information Processing Systems 32*, pages 393–403.

Chen, J., Zhu, J., Teh, Y. W., and Zhang, T. (2018). Stochastic expectation maximization with variance reduction. In *Advances in Neural Information Processing Systems 31*, pages 7967–7977.

Collins, M., Globerson, A., Koo, T., Carreras, X., and Bartlett, P. L. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, 9(Aug):1775–1822.

Combettes, P. L. and Pesquet, J.-C. (2011). Proximal splitting methods in signal processing. In *Fixed-point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer.

Csiba, D., Qu, Z., and Richtárik, P. (2015). Stochastic dual coordinate ascent with adaptive probabilities. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 674–683.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*.

Defazio, A. and Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. *Advances in Neural Information Processing Systems 33*.

Fang, C., Li, C. J., Lin, Z., and Zhang, T. (2018). Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems 31*, pages 689–699.

Fletcher, R. (1970). A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–323.

Frostig, R., Ge, R., Kakade, S. M., and Sidford, A. (2015). Competing with the empirical risk minimizer in a single pass. In *Conference on Learning Theory*, pages 728–763.

Garber, D. and Hazan, E. (2015). Fast and simple PCA via convex optimization. *CoRR*, abs/1509.05647.

Gazagnadou, N., Gower, R. M., and Salmon, J. (2019). Optimal mini-batch and step sizes for SAGA. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2142–2150.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26.

Gong, P. and Ye, J. (2014). Linear convergence of variance-reduced stochastic gradient without strong convexity. *arXiv:1406.1102*.

Gorbunov, E., Hanzely, F., and Richtárik, P. (2019). A unified theory of SGD: Variance reduction, sampling, quantization and coordinate descent. *arXiv:1905.11261*.

Gower, R., Goldfarb, D., and Richtarik, P. (2016). Stochastic block BFGS: Squeezing more curvature out of data. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1869–1878.

Gower, R. M., Loizou, N., Qian, X., Sailanbayev, A., Shulgin, E., and Richtárik, P. (2019). SGD: General Analysis and Improved Rates. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 5200–5209.

Gower, R. M., Richtárik, P., and Bach, F. (2018). Stochastic quasi-gradient methods: Variance reduction via Jacobian sketching. *arXiv:1805.02632*.

Harikandeh, R., Ahmed, M. O., Virani, A., Schmidt, M., Konečný, J., and Sallinen, S. (2015). Stop wasting my gradients: Practical SVRG. In *Advances in Neural Information Processing Systems 28*, pages 2251–2259.

Hofmann, T., Lucchi, A., Lacoste-Julien, S., and McWilliams, B. (2015). Variance reduced stochastic gradient descent with neighbors. In *Neural Information Processing Systems 28*, pages 2305–2313.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pages 315–323.

Karimi, H., Nutini, J., and Schmidt, M. (2016). Linear convergence of gradient and proximal-gradient methods under the Polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*.

Konečný, J., Liu, J., Richtárik, P., and Takáč, M. (2016). Mini-batch semi-stochastic gradient descent in the proximal setting. *Journal of Selected Topics in Signal Processing*, pages 242–255.

Konecný, J. and Richtárik, P. (2013). Semi-stochastic gradient descent methods. *CoRR*, abs/1312.1666.

Kovalev, D., Horváth, S., and Richtárik, P. (2020). Don't jump through hoops and remove those loops: SVRG and Katyusha are better without the outer loop. In *Proceedings of the 31st International Conference on Algorithmic Learning Theory*.

Kovalev, D., Mishchenko, K., and Richtárik, P. (2019). Stochastic Newton and cubic Newton methods with simple local linear-quadratic rates. In *NeurIPS Beyond First Order Methods Workshop*.

Kulunchakov, A. and Mairal, J. (2019). Estimate sequences for variance-reduced stochastic composite optimization. In *International Conference on Machine Learning*, pages 3541–3550.

Lan, G. and Zhou, Y. (2018). An optimal randomized incremental gradient method. *Mathematical Programming*, 171(1-2):167–215.

Le Roux, N., Schmidt, M., and Bach, F. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. (2017). ASAGA: Asynchronous parallel SAGA. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Lei, L. and Jordan, M. (2017). Less than a single pass: Stochastically controlled stochastic gradient. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 148–156.

Lian, X., Wang, M., and Liu, J. (2017). Finite-sum composition optimization via variance reduced gradient descent. In *International Conference on Artificial Intelligence and Statistics*.

Lin, H., Mairal, J., and Harchaoui, Z. (2018). Catalyst acceleration for first-order convex optimization: from theory to practice. *Journal of Machine Learning Research (JMLR).*, 18(212):1–54.

Lohr, S. (1999). *Sampling: Design and Analysis*. Duxbury Press.

Mahdavi, M. and Jin, R. (2013). MixedGrad: An O(1/T) convergence rate algorithm for stochastic smooth optimization. *arXiv:1307.7192*.

Mairal, J. (2015). Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855.

Malitsky, Y. and Mishchenko, K. (2019). Adaptive gradient descent without descent. *arXiv:1910.09529*.

Mokhtari, A. and Ribeiro, A. (2014). Regularized stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, 62:1109–1112.

Mokhtari, A. and Ribeiro, A. (2015). Global convergence of online limited memory BFGS. *The Journal of Machine Learning Research*, 16:3151–3181.

Moritz, P., Nishihara, R., and Jordan, M. I. (2016). A linearly-convergent stochastic L-BFGS algorithm. In *International Conference on Artificial Intelligence and Statistics*, pages 249–258.

Needell, D., Srebro, N., and Ward, R. (2015). Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Mathematical Programming*, 155(1):549–573.

Nesterov, Y. (1983). A method for solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376.

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

Nesterov, Y. (2014). *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition.

Nesterov, Y. and Polyak, T. B. (2006). Cubic regularization of newton method and its global performance. *Mathematical Programming*, pages 177–205.

Nguyen, L. M., Liu, J., Scheinberg, K., and Takáč, M. (2017). SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2613–2621.

Palaniappan, B. and Bach, F. (2016). Stochastic variance reduction methods for saddle-point problems. In *Advances in Neural Information Processing Systems*, pages 1416–1424.

Papini, M., Binaghi, D., Canonaco, G., Pirotta, M., and Restelli, M. (2018). Stochastic variance-reduced policy gradient. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4026–4035.

Polyak, B. (1963). Gradient methods for the minimisation of functionals. *Ussr Computational Mathematics and Mathematical Physics*, 3:864–878.

Qian, X., Qu, Z., and Richtárik, P. (2019). SAGA with arbitrary sampling. In *The 36th International Conference on Machine Learning*.

Qu, Z., Richtárik, P., Takáč, M., and Fercoq, O. (2016). SDNA: Stochastic dual Newton ascent for empirical risk minimization. In *The 33rd International Conference on Machine Learning*, pages 1823–1832.

Qu, Z., Richtárik, P., and Zhang, T. (2015). Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 865–873.

Reddi, S. J., Hefny, A., Sra, S., Póczos, B., and Smola, A. (2016a). Stochastic variance reduction for nonconvex optimization. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 314–323.

Reddi, S. J., Sra, S., Póczos, B., and Smola, A. (2016b). Fast incremental method for smooth nonconvex optimization. In *2016 IEEE 55th Conference on Decision and Control*, pages 1971–1977.

Richtárik, P. and Takáč, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(2):1–38.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Roosta-Khorasani, F. and Mahoney, M. W. (2019). Sub-sampled Newton methods. *Mathematical Programmming*, 174(1–2).

Schmidt, M., Le Roux, N., and Bach, F. (2017). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1):83–112.

Schmidt, M. W., Babanezhad, R., Ahmed, M. O., Defazio, A., Clifton, A., and Sarkar, A. (2015). Non-uniform stochastic average gradient method for training conditional random fields. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS*.

Schraudolph, N. N. and Simon, G. (2007). A stochastic quasi-Newton method for online convex optimization. *In Proceedings of 11th International Conference on Artificial Intelligence and Statistics*.

Sebbouh, O., Gazagnadou, N., Jelassi, S., Bach, F., and Gower, R. M. (2019). Towards closing the gap between the theory and practice of SVRG. In *Advances in Neural Information Processing Systems 32*, pages 646–656.

Shalev-Shwartz, S. (2016). SDCA without duality, regularization, and individual convexity. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 747–754.

Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599.

Shalev-Shwartz, S. and Zhang, T. (2014). Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *International Conference on Machine Learning*, pages 64–72.

Shamir, O. (2015). A stochastic PCA and SVD algorithm with an exponential convergence rate. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, volume 37, pages 144–152.

Shanno, D. F. (1971). Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656.

Strohmer, T. and Vershynin, R. (2009). A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262.

Takáč, M., Bijral, A., Richtárik, P., and Srebro, N. (2013). Mini-batch primal and dual methods for SVMs. In *30th International Conference on Machine Learning*, pages 537–552.

Vainsencher, D., Liu, H., and Zhang, T. (2015). Local smoothness in variance reduced optimization. In *Advances in Neural Information Processing Systems*, pages 2179–2187.

Wai, H.-T., Hong, M., Yang, Z., Wang, Z., and Tang, K. (2019). Variance reduced policy evaluation with smooth function approximation. In *Advances in Neural Information Processing Systems 32*, pages 5776–5787.

Wang, X., Ma, S., Goldfarb, D., and Liu, W. (2017). Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM Journal of Optimization*, 27:927–956.

Wang, Z., Zhou, Y., Liang, Y., and Lan, G. (2018). Sample complexity of stochastic variance-reduced cubic regularization for nonconvex optimization. *CoRR*, abs/1802.07372.

Woodworth, B. E. and Srebro, N. (2016). Tight complexity bounds for optimizing composite objectives. In *Advances in Neural Information Processing Systems 29*, pages 3639–3647.

Xiao, L. and Zhang, T. (2014). A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075.

Xu, P., Gao, F., and Gu, Q. (2019). An improved convergence analysis of stochastic variance-reduced policy gradient. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, page 191.

Zhang, J. and Xiao, L. (2020). Stochastic variance-reduced prox-linear algorithms for nonconvex composite optimization. Technical report, Microsoft.

Zhang, L., Mahdavi, M., and Jin, R. (2013). Linear convergence with condition number independent access of full gradients. In *Advances in Neural Information Processing Systems*, pages 980–988.

Zhang, Y. and Xiao, L. (2017). Stochastic primal-dual coordinate method for regularized empirical risk minimization. *The Journal of Machine Learning Research*, 18(1):2939–2980.

Zhong, L. W. and Kwok, J. T. (2014). Fast stochastic alternating direction method of multipliers. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 46–54.

Zhou, D. and Gu, Q. (2019). Lower bounds for smooth nonconvex finite-sum optimization. *International Conference on Machine Learning*, pages 7574–7583.

Zhou, D., Xu, P., and Gu, Q. (2018). Stochastic nested variance reduced gradient descent for nonconvex optimization. In *Advances in Neural Information Processing Systems 31*, pages 3921–3932.

Zhou, D., Xu, P., and Gu, Q. (2019). Stochastic variance-reduced cubic regularization methods. *Journal of Machine Learning Research*, 20(134):1–47.

Zou, D., Xu, P., and Gu, Q. (2019). Stochastic gradient hamiltonian monte carlo methods with recursive variance reduction. In *Advances in Neural Information Processing Systems 32*, pages 3830–3841.

## A. Lemmas

Here we provide and prove several auxiliary lemmas.

**Lemma A.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*

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

*Proof.* Since $f_i$ is convex we have that

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

and since $f_i$ is smooth we have that

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

see Section 2.1.1 in Nesterov (2014) for more equivalent ways of re-writing convexity and smoothness. 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

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

Substituting this in the above gives

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

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

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

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

*Proof.*

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

■

## 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

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

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 $\mu$-strongly convex, we have

$$\begin{aligned}\mathbf{E}_k\left[\|x_{k+1}-x_\star\|^2\right] &\leq (1-\mu\gamma)\|x_k-x_\star\|^2 + \gamma^2\mathbf{E}_k\left[\|g_k\|^2\right] \\ &\quad - 2\gamma(f(x_k)-f(x_\star)).\end{aligned} \quad [40]$$

To conclude the proof, we need a bound on the second moment $\mathbf{E}_k\left[\|g_k\|^2\right]$ of $g_k$. For the plain vanilla SGD, often it is simply assumed that this variance term is bounded uniformly by an unknown constant $B>0$. But 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

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

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

**Theorem B.1 (?).** *Consider the iterates of* SGD$_\star$ [13]. *If Assumptions 1.1 and 1.2 hold and* $\gamma \leq \frac{1}{L_{\max}}$, *then the iterates converge linearly with*

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

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

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

*Proof.* Using Lemma A.1, we have

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

Using the above in [40] we have

$$\begin{aligned}\mathbf{E}_k\left[\|x_{k+1}-x_\star\|^2\right] &\leq (1-\mu\gamma)\|x_k-x_\star\|^2 \\ &\quad + 2\gamma(\gamma L_{\max}-1)(f(x_k)-f(x_\star)).\end{aligned} \quad [45]$$

Now by choosing $\gamma \leq \frac{1}{L_{\max}}$ 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 $f(x_k)-f(x_\star)\geq 0$. So it now follows by taking expectation in [45] that

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

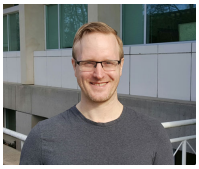
■

This proof also shows that the shifted SGD method is a variance-reduced 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 A.2 with $X = \nabla f_i(x_k) - \nabla f_i(x_\star)$.

**Robert M. Gower** is a visiting researcher at Facebook AI Research (2020 New York) and joined Télécom Paris 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. A mathematician by training, his academic studies started with a Bachelors and a Masters degree in applied mathematics at the state University of Campinas (Brazil), where he designed the current state-of-the-art algorithms for automatically calculating high order derivatives using back-propagation.



**Mark Schmidt** is an associate professor in the Department of Computer Science at the University of British Columbia. His research focuses on machine learning and numerical optimization. He is a Canada Research Chair, Alfred P. Sloan Fellow, 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** is a researcher at Inria, leading since 2011 the machine learning team which is part of the Computer Science department at École Normale Supérieure. He graduated from École Polytechnique in 1997 and completed his Ph.D. in Computer Science at U.C. Berkeley in 2005, working with Professor Michael Jordan. He spent two years in the Mathematical Morphology group at École des Mines de Paris, then he joined the computer vision project-team at Inria/École Normale Supérieure from 2007 to 2010. Francis Bach is primarily interested in machine learning, and especially in sparse methods, kernel-based learning, large-scale optimization, computer vision and signal processing. He obtained in 2009 a Starting Grant and in 2016 a Consolidator Grant from the European Research Council, and received the Inria young researcher prize in 2012, the ICML test-of-time

award in 2014, as well as the Lagrange prize in continuous optimization in 2018, and the Jean-Jacques Moreau prize in 2019. In 2015, he was program co-chair of the International Conference in Machine learning (ICML), and general chair in 2018. He is now co-editor-in-chief of the Journal of Machine Learning Research.

**Peter Richtárik** is a Professor of Computer Science and Mathematics at KAUST. He is an EPSRC Fellow in Mathematical Sciences, Fellow of the Alan Turing Institute, and is affiliated with the Visual Computing Center and the Extreme Computing Research Center at KAUST. Prof. Richtárik received his PhD from Cornell University in 2007, and then worked as a Postdoctoral Fellow in Louvain, Belgium, before joining Edinburgh in 2009, and KAUST in 2017. 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. Several of his papers attracted international awards to his collaborators, including the SIAM SIGEST Best Paper Award, the IMA Leslie Fox Prize (2nd prize, three times), and the INFORMS Computing Society Best Student Paper Award (sole runner up).