Math. Program., Ser. A (2017) 162:83–112
DOI 10.1007/s10107-016-1030-6

FULL LENGTH PAPER

Minimizing finite sums with the stochastic average gradient

Mark Schmidt[1] · **Nicolas Le Roux**[2] · **Francis Bach**[3]

Received: 11 September 2013 / Accepted: 13 May 2016 / Published online: 14 June 2016

Abstract We analyze the stochastic average gradient (SAG) method for optimizing the sum of a finite number of smooth convex functions. Like stochastic gradient (SG) methods, the SAG method's iteration cost is independent of the number of terms in the sum. However, by incorporating a memory of previous gradient values the SAG method achieves a faster convergence rate than black-box SG methods. The convergence rate is improved from $O(1/\sqrt{k})$ to $O(1/k)$ in general, and when the sum is strongly-convex the convergence rate is improved from the sub-linear $O(1/k)$ to a linear convergence rate of the form $O(\rho^k)$ for $\rho < 1$. Further, in many cases the convergence rate of the new method is also faster than black-box deterministic gradient methods, in terms of the number of gradient evaluations. This extends our earlier work Le Roux et al. (Adv Neural Inf Process Syst, 2012), which only lead to a faster rate for well-conditioned strongly-convex problems. Numerical experiments indicate that the new algorithm often dramatically outperforms existing SG and deterministic

Electronic supplementary material The online version of this article (doi:10.1007/s10107-016-1030-6) contains supplementary material, which is available to authorized users.

✉ Mark Schmidt
schmidtm@cs.ubc.ca

Nicolas Le Roux
nicolas@le-roux.name

Francis Bach
francis.bach@ens.fr

[1] Department of Computer Science, University of British Columbia, 201 2366 Main Mall, Vancouver, BC V6T 1Z4, Canada

[2] Criteo, 32 rue Blanche, 75009 Paris, France

[3] Laboratoire d'Informatique de l'Ecole Normale Superieure, INRIA - SIERRA Project-Team, 23, avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France

gradient methods, and that the performance may be further improved through the use of non-uniform sampling strategies.

Keywords Convex optimization · Stochastic gradient methods · First-order methods · Convergence Rates

Mathematics Subject Classification 90C06 · 90C15 · 90C25 · 90C30 · 65K05 · 68Q25 · 62L20

1 Introduction

A plethora of the optimization problems arising in practice involve computing a minimizer of a finite sum of functions measuring misfit over a large number of data points. A classical example is least-squares regression,

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}(a_i^T x - b_i)^2,$$

where the $a_i \in \mathbb{R}^p$ and $b_i \in \mathbb{R}$ are the data samples associated with a regression problem. Another important example is logistic regression,

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\quad \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-b_i a_i^\top x)),$$

where the $a_i \in \mathbb{R}^p$ and $b_i \in \{-1, 1\}$ are the data samples associated with a binary classification problem. A key challenge arising in modern applications is that the number of data points n (also known as *training examples*) can be extremely large, while there is often a large amount of redundancy between examples. The most wildly successful class of algorithms for taking advantage of the *sum* structure for problems where n is very large are *stochastic gradient* (SG) methods [7,49]. Although the theory behind SG methods allows them to be applied more generally, SG methods are often used to solve the problem of optimizing a finite sample average,

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\quad g(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{1}$$

In this work, we focus on such *finite data* problems where each f_i is *smooth* and *convex*.

In addition to this basic setting, we will also be interested in cases where the sum g has the additional property that it is *strongly-convex*. This often arises due to the use of a strongly-convex regularizer such as the squared ℓ_2-norm, resulting in problems of the form

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\quad \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} l_i(x), \tag{2}$$

where each l_i is a data-misfit function (as in least-squares and logistic regression) and the positive scalar λ controls the strength of the regularization. These problems can be put in the framework of (1) by using the choice

$$f_i(x) := \frac{\lambda}{2}\|x\|^2 + l_i(x).$$

The resulting function g will be strongly-convex provided that the individual loss functions l_i are convex. An extensive list of convex loss functions used in a statistical data-fitting context is given by Teo et al. [61], and non-smooth loss functions (or regularizers) can also be put in this framework by using smooth approximations (for example, see [43]).

For optimizing problem (1), the standard *deterministic* or *full gradient* (FG) method, which dates back to Cauchy [10], uses iterations of the form

$$x^{k+1} = x^k - \alpha_k g'(x^k) = x^k - \frac{\alpha_k}{n}\sum_{i=1}^{n} f_i'(x^k), \tag{3}$$

where α_k is the step size on iteration k. Assuming that a minimizer x^* exists, then under standard assumptions the sub-optimality achieved on iteration k of the FG method with a constant step size is given by

$$g(x^k) - g(x^*) = O(1/k),$$

when g is convex [42, see Corollary 2.1.2]. This results in a *sublinear* convergence rate. When g is strongly-convex, the error also satisfies

$$g(x^k) - g(x^*) = O(\rho^k),$$

for some $\rho < 1$ which depends on the condition number of g [42, see Theorem 2.1.5]. This results in a *linear* convergence rate, which is also known as a *geometric* or *exponential* rate because the error is cut by a fixed fraction on each iteration. Unfortunately, the FG method can be unappealing when n is large because its iteration cost scales linearly in n.

The main appeal of SG methods is that they have an iteration cost which is *independent* of n, making them suited for modern problems where n may be very large. The basic SG method for optimizing (1) uses iterations of the form

$$x^{k+1} = x^k - \alpha_k f_{i_k}'(x^k), \tag{4}$$

where at each iteration an index i_k is sampled uniformly from the set $\{1, \ldots, n\}$. The randomly chosen gradient $f_{i_k}'(x^k)$ yields an unbiased estimate of the true gradient $g'(x^k)$ and one can show under standard assumptions (see [40]) that, for a suitably

chosen decreasing step-size sequence $\{\alpha_k\}$, the SG iterations have an expected sub-optimality for convex objectives of

$$\mathbb{E}[g(x^k)] - g(x^*) = O(1/\sqrt{k}),$$

and an expected sub-optimality for strongly-convex objectives of

$$\mathbb{E}[g(x^k)] - g(x^*) = O(1/k).$$

In these rates, the expectations are taken with respect to the selection of the i_k variables. These sublinear rates are slower than the corresponding rates for the FG method, and under certain assumptions these convergence rates are *optimal* in a model of computation where the algorithm only accesses the function through unbiased measurements of its objective and gradient (see [1,39,40]). Thus, we should not expect to be able to obtain the convergence rates of the FG method if the algorithm only relies on unbiased gradient measurements. Nevertheless, by using the stronger assumption that the functions are sampled from a finite dataset, in this paper we show that we can achieve the convergence rates of FG methods while preserving the iteration complexity of SG methods.

The primary contribution of this work is the analysis of a new algorithm that we call the *stochastic average gradient* (SAG) method, a randomized variant of the incremental aggregated gradient (IAG) method of Blatt et al. [5]. The SAG method has the low iteration cost of SG methods, but achieves the convergence rates stated above for the FG method. The SAG iterations take the form

$$x^{k+1} = x^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} y_i^k, \tag{5}$$

where at each iteration a random index i_k is selected and we set

$$y_i^k = \begin{cases} f_i'(x^k) & \text{if } i = i_k, \\ y_i^{k-1} & \text{otherwise.} \end{cases} \tag{6}$$

That is, like the FG method, the step incorporates a gradient with respect to each function. But, like the SG method, each iteration only computes the gradient with respect to a single example and the cost of the iterations is independent of n. Despite the low cost of the SAG iterations, we show in this paper that with a constant step-size *the SAG iterations have an* $O(1/k)$ *convergence rate for convex objectives and a linear convergence rate for strongly-convex objectives*, like the FG method. That is, by having access to i_k and by keeping a *memory* of the most recent gradient value computed for each index i, this iteration achieves a faster convergence rate than is possible for standard SG methods. Further, in terms of effective passes through the data, we will also see that for many problems the convergence rate of the SAG method is also faster than is possible for standard FG methods.

One of the main contexts where minimizing the sum of smooth convex functions arises is machine learning. In this context, g is often an *empirical* risk (or a regularized empirical risk), which is a sample average approximation to the *true* risk that we are interested in. It is known that with n training examples the empirical risk minimizer (ERM) has an error for the true risk of $O(1/\sqrt{n})$ in the convex case and $O(1/n)$ in the strongly-convex case. Since these rates are achieved by doing one pass through the data with an SG method, in the worst case the SAG algorithm applied to the empirical risk cannot improve the convergence rate in terms of the true risk over this simple method. Nevertheless, Srebro and Sridharan [58] note that "overwhelming empirical evidence shows that for almost all actual data, the ERM *is* better. However, we have no understanding of why this happens". Although our analysis does not give insight into the better performance of ERM, our analysis shows that the SAG algorithm will be preferable to SG methods for finding the ERM and hence for many machine learning applications.

The next section reviews several closely-related algorithms from the literature, including previous attempts to combine the appealing aspects of FG and SG methods. However, despite 60 years of extensive research on SG methods, with a significant portion of the applications focusing on finite datasets, we believe that this is the first general method that achieves the convergence rates of FG methods while preserving the iteration cost of standard SG methods. Section 3 states the (standard) assumptions underlying our analysis and gives our convergence rate results. Section 4 discusses practical implementation issues including how we adaptively set the step size and how we can reduce the storage cost needed by the algorithm. For example, we can reduce the memory requirements from $O(np)$ to $O(n)$ in the common scenario where each f_i only depends on a linear function of x, as in least-squares and logistic regression. Section 5 presents a numerical comparison of an implementation based on SAG to competitive SG and FG methods, indicating that the method may be very useful for problems where we can only afford to do a few passes through a data set.

A preliminary conference version of this work appears in [30], and we extend this work in various ways. Most notably, the analysis in the prior work focuses only on showing linear convergence rates in the strongly-convex case while the present work also gives an $O(1/k)$ convergence rate for the general convex case. In the prior work we show (Proposition 1) that a small step-size gives a slow linear convergence rate (comparable to the rate of FG methods in terms of effective passes through the data), while we also show (Proposition 2) that a much larger step-size yields a much faster convergence rate, but this requires that n is sufficiently large compared to the condition number of the problem. In the present work (Sect. 3) our analysis yields a very fast convergence rate using a large step-size (Theorem 1), even when this condition required by the prior work is not satisfied. Surprisingly, for ill-conditioned problems our new analysis shows that using SAG iterations can be nearly n times as fast as the standard gradient method. To prove this stronger result, Theorem 1 employs a Lyapunov function that generalizes the Lyapunov functions used in Propositions 1 and 2 of the previous work. This new Lyapunov function leads to a unified proof for both the convex and the strongly-convex cases, and for both well-conditioned and ill-conditioned problems. However, this more general Lyapunov function leads to a more complicated analysis. To significantly simplify the formal proof, we use

a computed-aided strategy to verify the non-negativity of certain polynomials that arise in the proof. Beyond this significantly strengthened result, in this work we also argue that yet-faster convergence rates may be achieved by *non-uniform* sampling (Sect. 4.8) and present numerical results showing that this can lead to drastically improved performance (Sect. 5.3).

Due to space restrictions, some details are omitted in this article. This notably includes the proof of the main theorem, some additional experimental results, and a thorough discussion of the many interesting works that have followed after [30]. These extra materials are made available in the extrended arXiv version of the paper located at: http://arxiv.org/abs/1309.2388.

2 Related work

There are a large variety of approaches available to accelerate the convergence of SG methods, and a full review of this immense literature would be outside the scope of this work. Below, we comment on the relationships between the new method and several of the most closely-related ideas.

2.1 Momentum

SG methods that incorporate a momentum term use iterations of the form

$$x^{k+1} = x^k - \alpha_k f'_{i_k}(x^k) + \beta_k (x^k - x^{k-1}),$$

see [62]. It is common to set all $\beta_k = \beta$ for some constant β, and in this case we can rewrite the SG with momentum method as

$$x^{k+1} = x^k - \sum_{j=1}^{k} \alpha_j \beta^{k-j} f'_{i_j}(x^j).$$

We can re-write the SAG updates (5) in a similar form as

$$x^{k+1} = x^k - \sum_{j=1}^{k} \alpha_k S(j, i_{1:k}) f'_{i_j}(x^j), \tag{7}$$

where the selection function $S(j, i_{1:k})$ is equal to $1/n$ if j is the maximum iteration number where example i_j was selected and is set to 0 otherwise. Thus, momentum uses a *geometric weighting* of previous gradients while the SAG iterations *select and average* the most recent evaluation of each previous gradient. While momentum can lead to improved practical performance, it still requires the use of a decreasing sequence of step sizes and is not known to lead to a faster convergence rate.

2.2 Gradient averaging

Closely related to momentum is using the sample average of all previous gradients,

$$x^{k+1} = x^k - \tfrac{\alpha_k}{k} \sum_{j=1}^{k} f'_{i_j}(x_j),$$

which is similar to the SAG iteration in the form (5) but where *all* previous gradients are used. This approach is used in the dual averaging method of Nesterov [44] and, while this averaging procedure and its variants lead to convergence for a constant step size and can improve the constants in the convergence rate [63], it does not improve on the sublinear convergence rates for SG methods.

2.3 Iterate averaging

Rather than averaging the gradients, some authors propose to perform the basic SG iteration but use an average over certain x^k values as the final estimator. With a suitable choice of step-sizes, this gives the same asymptotic efficiency as Newton-like second-order SG methods and also leads to increased robustness of the convergence rate to the exact sequence of step sizes [2,47]. Bather's method [28, § 1.3.4] combines gradient averaging with online iterate averaging and also displays appealing asymptotic properties. Several authors have recently shown that suitable iterate averaging schemes obtain an $O(1/k)$ rate for strongly-convex optimization even for non-smooth objectives [21,48]. However, none of these methods improve on the $O(1/\sqrt{k})$ and $O(1/k)$ rates for SG methods.

2.4 Stochastic versions of FG methods

Various options are available to accelerate the convergence of the FG method for smooth functions, such as the accelerated full gradient (AFG) method of Nesterov [41], as well as classical techniques based on quadratic approximations such as diagonally-scaled FG methods, non-linear conjugate gradient, quasi-Newton, and Hessian-free Newton methods (see [46]). There has been a substantial amount of work on developing stochastic variants of these algorithms, with several of the notable recent examples including [6,18,22,36,60,63]. Duchi et al. [15] have recently shown an improved regret bound using a diagonal scaling that takes into account previous gradient magnitudes. Alternately, if we split the convergence rate into a deterministic and stochastic part, these methods can improve the dependency of the convergence rate of the deterministic part [18,22,63]. However, we are not aware of any existing method of this flavor that improves on the $O(1/\sqrt{k})$ and $O(1/k)$ dependencies on the stochastic part. Further, many of these methods typically require carefully setting parameters (beyond the step size) and often aren't able to take advantage of sparsity in the gradients f'_i.

2.5 Constant step size

If the SG iterations are used for strongly-convex optimization with a *constant* step size (rather than a decreasing sequence), then Nedic and Bertsekas [37, Proposition 3.4] showed that the convergence rate of the method can be split into two parts. The first part depends on k and converges linearly to 0. The second part is independent of k and does not converge to 0. Thus, with a constant step size, the SG iterations have a linear convergence rate up to some tolerance, and in general after this point the iterations do not make further progress. Indeed, up until the recent work of Bach and Moulines [3], convergence of the basic SG method with a constant step size had only been shown for the strongly-convex quadratic case (with averaging of the iterates) [47], or under extremely strong assumptions about the relationship between the functions f_i [57]. This contrasts with the method we present in this work which converges to the optimal solution using a constant step size *and* does so with a linear rate (without additional assumptions).

2.6 Accelerated methods

Accelerated SG methods, which despite their name are not related to the aforementioned AFG method, take advantage of the fast convergence rate of SG methods with a constant step size. In particular, accelerated SG methods use a constant step size by default, and only decrease the step size on iterations where the inner-product between successive gradient estimates is negative [14,25]. This leads to convergence of the method and allows it to potentially achieve periods of faster convergence where the step size stays constant. However, the overall convergence rate of the method is not improved.

2.7 Hybrid methods

Some authors have proposed variants of the SG method for problems of the form (1) that seek to gradually transform the iterates into the FG method in order to achieve a faster convergence rate. Bertsekas [4] proposes to go through the data cyclically with a specialized weighting that allows the method to achieve a linear convergence rate for strongly-convex quadratic functions. However, the weighting is numerically unstable and the linear convergence rate presented treats full passes through the data as iterations. A related strategy is to group the functions f_i into 'batches' of increasing size and perform SG iterations on the batches. Friedlander and Schmidt [17] give conditions under which this strategy achieves the $O(1/k)$ and $O(\rho^k)$ convergence rates of FG methods. However, in both cases the iterations that achieve the faster rates have a cost that is not independent of n, as opposed to SAG iterations.

2.8 Incremental aggregated gradient

Blatt et al. [5] present the most closely-related algorithm to the SAG algorithm, the IAG method. This method is identical to the SAG iteration (5), but uses a cyclic

choice of i_k rather than sampling the i_k values. This distinction has several important consequences. In particular, Blatt et al. are only able to show that the convergence rate is linear for strongly-convex quadratic functions (without deriving an explicit rate), and their analysis treats full passes through the data as iterations. Using a non-trivial extension of their analysis and a novel proof technique involving bounding the gradient and iterates simultaneously by a Lyapunov potential function, in this work *we give an* $O(1/k)$ *rate for general convex functions and an explicit linear convergence rate for general strongly-convex functions using the SAG iterations that only examine a single function*. Further, as our analysis and experiments show, the SAG iterations allow a much larger step size than is required for convergence of the IAG method. This leads to more robustness to the selection of the step size and also, if suitably chosen, leads to a faster convergence rate and substantially improved practical performance. This shows that the simple change (random selection vs. cycling) can dramatically improve optimization performance.

2.9 Special problem classes

For certain highly-restricted classes of problems, it is possible to show faster convergence rates for methods that only operate on a single function f_i. For example, Strohmer and Vershynin [59] show that the randomized Kaczmarz method with a particular sampling scheme achieves a linear convergence rate for the problem of solving consistent linear systems. It is also known that the SG method with a constant step-size has the $O(1/k)$ and $O(\rho^k)$ convergence rates of FG methods if $\|f_i'(x)\|$ is bounded by a linear function of $\|g'(x)\|$ for all i and x [52]. This is the strong condition required by Solodov [57] to show convergence of the SG method with a constant step size. Unlike these previous works, our analysis in the next section applies to general f_i that satisfy standard assumptions, and only requires gradient evaluations of the functions f_i rather than dual block-coordinate steps.

2.10 Subsequent work

Since the first version of this work was released, there has been an explosion of research into stochastic gradient methods with faster convergence rates. It has been shown that similar rates can be achieved for certain constrained and non-smooth problems, that similar rates can be achieved without the memory requirements, that Newton-like variants of the method may be possible, and that similar rates can be achieved with other algorithms. In Sect. 6 of the extended version of this paper, we survey these recent developments.

3 Convergence analysis

In our analysis we assume that each function f_i in (1) is convex and differentiable, and that each gradient f_i' is Lipschitz-continuous with constant L, meaning that for all x and y in $\mathbb{R}^p$ and each i we have

$$\|f_i'(x) - f_i'(y)\| \leqslant L\|x - y\|. \tag{8}$$

This is a fairly weak assumption on the f_i functions, and in cases where the f_i are twice-differentiable it is equivalent to saying that the eigenvalues of the Hessians of each f_i are bounded above by L. We will also assume the existence of at least one minimizer x^* that achieves the optimal function value. We denote the average iterate by $\bar{x}^k = \frac{1}{k}\sum_{i=0}^{k-1} x^i$, and the variance of the gradient norms at the optimum x^* by $\sigma^2 = \frac{1}{n}\sum_i \|f_i'(x^*)\|^2$. Our convergence results consider two different initializations for the y_i^0 variables: setting $y_i^0 = 0$ for all i, or setting them to the centered gradient at the initial point x^0 given by $y_i^0 = f_i'(x^0) - g'(x^0)$. We note that all our convergence results are expressed in terms of expectations with respect to the internal randomization of the algorithm (the selection of the random variables i_k), and not with respect to the data which is assumed to be deterministic and fixed.

In addition to this basic convex case discussed above, we will also consider the case where the average function $g = \frac{1}{n}\sum_{i=1}^n f_i$ is strongly-convex with constant $\mu > 0$, meaning that the function $x \mapsto g(x) - \frac{\mu}{2}\|x\|^2$ is convex. For twice-differentiable g, this is equivalent to requiring that the eigenvalues of the Hessian of g are bounded below by μ. This is a stronger assumption that is often not satisfied in practical applications. Nevertheless, in many applications we are free to choose a regularizer of the parameters, and thus we can add an ℓ_2-regularization term as in (2) to transform any convex problem into a strongly-convex problem (in this case we have $\mu \geq \lambda$). Note that strong-convexity implies the existence of a unique x^* that achieves the optimal function value.

Under these standard assumptions, we now state our convergence result.

Theorem 1 *With a constant step size of $\alpha_k = \frac{1}{16L}$, the SAG iterations satisfy for $k \geq 1$:*

$$\mathbb{E}[g(\bar{x}^k)] - g(x^*) \leqslant \frac{32n}{k}C_0,$$

where if we initialize with $y_i^0 = 0$ we have

$$C_0 = g(x^0) - g(x^*) + \frac{4L}{n}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L},$$

and if we initialize with $y_i^0 = f_i'(x^0) - g'(x^0)$ we have

$$C_0 = \frac{3}{2}\Big[g(x^0) - g(x^*)\Big] + \frac{4L}{n}\|x^0 - x^*\|^2.$$

Further, if g is μ-strongly convex we have

$$\mathbb{E}[g(x^k)] - g(x^*) \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^k C_0.$$

The proof is given in Appendix B of the extended version of this paper, and involves finding a Lyapunov function for a non-linear stochastic dynamical system defined on the y_i^k and x^k variables that converges to zero at the above rates, and showing that this function dominates the expected sub-optimality $[\mathbb{E}[g(x^k)] - g(x^*)]$. This is the same approach used to show Proposition 1 and 2 in the conference version of the paper [30], but in this work we use a more general Lyapunov function that gives a much faster rate for ill-conditioned problems and also allows us to analyze problems that are not strongly-convex. To simplify the analysis of this more complicated Lyapunov function, our new proof verifies positivity of certain polynomials that arise in the bound using a computer-aided approach.

Note that while the first part of Theorem 1 is stated for the average $\bar{x}^k$, with a trivial change to the proof technique it can be shown to also hold for any iterate x^k where $g(x^k)$ is lower than the average function value up to iteration k, $\frac{1}{k}\sum_{i=0}^{k-1} g(x^i)$. Thus, in addition to $\bar{x}^k$ the result also holds for the best iterate. We also note that our bounds are valid for any L greater than or equal to the minimum L satisfying (8), implying an $O(1/k)$ and linear convergence rate for any $\alpha_k \leqslant 1/16L$ (but the bound becomes worse as L grows). Although initializing each y_i^0 with the centered gradient may have an additional cost and slightly worsens the dependency on the initial sub-optimality $(g(x^0) - g(x^*))$, it removes the dependency on the variance σ^2 of the gradients at the optimum. While we have stated Theorem 1 in terms of the function values, in the strongly-convex case we also obtain a convergence rate on the iterates because we have

$$\frac{\mu}{2}\|x^k - x^*\|^2 \leqslant g(x^k) - g(x^*).$$

Theorem 1 shows that the SAG iterations are advantageous over SG methods in later iterations because they obtain a faster convergence rate. However, the SAG iterations have a worse constant factor because of the dependence on n. We can improve the dependence on n using an appropriate choice of x^0. In particular, following [30] we can set x^0 to the result of n iterations of an appropriate SG method. In this setting, the expectation of $g(x^0) - g(x^*)$ is $O(1/\sqrt{n})$ in the convex setting, while both $g(x^0) - g(x^*)$ and $\|x^0 - x^*\|^2$ would be in $O(1/n)$ in the strongly-convex setting. If we use this initialization of x^0 and set $y_i^0 = f_i'(x^0) - g'(x^0)$, then in terms of n and k the SAG convergence rates take the form $O(\sqrt{n}/k)$ and $O(\rho^k/n)$ in the convex and strongly-convex settings, instead of the $O(n/k)$ and $O(\rho^k)$ rates implied by Theorem 1. However, in our experiments we do not use an SG initialization but rather use a minor variant of SAG in the early iterations (discussed in the next section), which appears more difficult to analyze but which gives better empirical performance.

An interesting consequence of using a step-size of $\alpha_k = 1/16L$ is that it makes the method *adaptive* to the strong-convexity constant μ. That is, for problems with a higher degree of *local* strong-convexity around the solution x^*, the algorithm will automatically take advantage of this and yield a faster local rate. This can even lead to a local linear convergence rate if the problem is strongly-convex near the optimum but not globally strongly-convex. This adaptivity to the problem difficulty is in contrast to SG methods whose sequence of step sizes typically depend on global constants and thus do not adapt to local strong-convexity.

We have observed in practice that the IAG method with a step size of $\alpha_k = \frac{1}{16L}$ may diverge. While the step-size needed for convergence of the IAG iterations is not precisely characterized, we have observed that it requires a step-size of approximately $1/nL$ in order to converge. Thus, the SAG iterations can tolerate a step size that is roughly n times larger, which leads to increased robustness to the selection of the step size. Further, as our analysis and experiments indicate, the ability to use a large step size leads to improved performance of the SAG iterations. Note that using randomized selection with a larger step-size leading to vastly improved performance is not an unprecedented phenomenon; the analysis of Nedic and Bertsekas [37] shows that the iterations of the basic stochastic gradient method with a constant step-size can achieve the same error bound as full cycles through the data of the cyclic variant of the method by using steps that are n times larger (see the discussion after Proposition 3.4). Related results also appear in [11,29] showing the advantage of stochastic optimization strategies over deterministic optimization strategies in the context of certain dual optimization problems.

The convergence rate of the SAG iterations in the strongly-convex case takes a somewhat surprising form. For ill-conditioned problems where $n \leqslant \frac{2L}{\mu}$, n does not appear in the convergence rate and *the SAG algorithm has nearly the same convergence rate as the FG method* with a step size of $1/16L$, even though it uses iterations which are n times cheaper. This indicates that the basic gradient method (under a slightly sub-optimal step-size) is not slowed down by using out-of-date gradient measurements for ill-conditioned problems. Although n appears in the convergence rate in the well-conditioned setting where $n > \frac{2L}{\mu}$, if we perform n iterations of SAG (i.e., one effective pass through the data), the error is multiplied by $(1 - 1/8n)^n \leqslant \exp(-1/8)$, which is independent of n. Thus, in this setting each pass through the data reduces the excess objective by a constant multiplicative factor that is independent of the problem.

It is interesting to compare the convergence rate of SAG in the strongly-convex case with the known convergence rates for first-order methods [42, see §2]. In Table 1, we use two examples to compare the convergence rate of SAG to the convergence rates of the standard FG method, the faster AFG method, and the lower-bound for any first-order strategy (under certain dimensionality assumptions) for optimizing a function g

Table 1 Comparison of convergence rates of first-order methods to the convergence rates of n iterations of SAG

Algorithm	Step size	Theoretical rate	Rate in case 1	Rate case 2
FG	$\frac{1}{L}$	$\left(1 - \frac{\mu}{L}\right)^2$	0.9998	1.000
FG	$\frac{2}{\mu+L}$	$\left(1 - \frac{2\mu}{L+\mu}\right)^2$	0.9996	1.000
AFG	$\frac{1}{L}$	$\left(1 - \sqrt{\frac{\mu}{L}}\right)$	0.9900	0.9990
Lower-bound	–	$\left(1 - \frac{2\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)^2$	0.9608	0.9960
SAG (n iters)	$\frac{1}{16L}$	$\left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^n$	0.8825	0.9938

In the examples we take $n = 100000$, $L = 100$, $\mu = 0.01$ (Case 1), and $\mu = 0.0001$ (Case 2)

satisfying our assumptions. In this table, we compare the rate obtained for these FG methods to the rate obtained by running n iterations of SAG, since this requires the same number of evaluations of f_i'. Case 1 in this table focuses on a well-conditioned case where the rate of SAG is $(1 - 1/8n)$, while Case 2 focuses on an ill-conditioned example where the rate is $(1 - \mu/16L)$. Note that in the latter case the $O(1/k)$ rate for the method may be faster.

In Table 1 we see that performing n iterations of SAG can actually lead to a rate that is faster than the lower bound for FG methods. Thus, for certain problems *SAG can be substantially faster than any FG method that does not use the structure of the problem*. However, we note that the comparison is somewhat problematic because L in the SAG rates is the Lipschitz constant of the f_i' functions, while in the FG method we only require that L is an upper bound on the Lipschitz continuity of g' so it may be much smaller. To give a concrete example that takes this into account and also considers the rates of dual methods and coordinate-wise methods, in Appendix A of the extended version of this paper we attempt to more carefully compare the rates obtained for SAG with the rates of primal and dual FG and coordinate-wise methods for the special case of ℓ_2-regularized least-squares regression.

4 Implementation details

In Algorithm 1 we give pseudo-code for an implementation of the basic method, where we use a variable d to track the quantity $(\sum_{i=1}^{n} y_i)$. This section focuses on further implementation details that are useful in practice. In particular, we discuss modifications that lead to better practical performance than the basic Algorithm 1, including ways to reduce the storage cost, how to handle regularization, how to set the step size, using mini-batches, and using non-uniform sampling. Note that an implementation of the algorithm that incorporates many of these aspects is available from the first author's webpage.

Algorithm 1 Basic SAG method for minimizing $\frac{1}{n}\sum_{i=1}^{n} f_i(x)$ with step size α.

$d = 0$, $y_i = 0$ for $i = 1, 2, \ldots, n$
for $k = 0, 1, \ldots$ **do**
  Sample i from $\{1, 2, \ldots, n\}$
  $d = d - y_i + f_i'(x)$
  $y_i = f_i'(x)$
  $x = x - \frac{\alpha}{n} d$
end for

4.1 Structured gradients and just-in-time parameter updates

For many problems the storage cost of $O(np)$ for the y_i^k vectors is prohibitive, but we can often use the structure of the gradients f_i' to reduce this cost. For example,

a commonly-used specialization of (1) is *linearly-parameterized* models which take form

$$\underset{x\in\mathbb{R}^p}{\text{minimize}} \quad g(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(a_i^\top x). \tag{9}$$

Since each a_i is constant, for these problems we only need to store the scalar $f'_{i_k}(u_i^k)$ for $u_i^k = a_{i_k}^\top x^k$ rather than the full gradient $a_i f'_i(u_i^k)$. This reduces the storage cost from $O(np)$ down to $O(n)$.

For problems where the vectors a_i are sparse, an individual gradient f'_i will inherit the sparsity pattern of the corresponding a_i. However, the update of x in Algorithm 1 appears unappealing since in general d will be dense, resulting in an iteration cost of $O(p)$. Nevertheless, we can take advantage of the simple form of the SAG updates to implement a 'just-in-time' variant of the SAG algorithm where the iteration cost is proportional to the number of non-zeroes in a_{i_k}. In particular, we do this by not explicitly storing the full vector x^k after each iteration. Instead, on each iteration we only compute the elements x_j^k corresponding to non-zero elements of a_{i_k}, by applying the *sequence of updates* to each variable x_j^k since the last iteration where it was non-zero in a_{i_k}. This sequence of updates can be applied efficiently since it simply involves changing the step size. For example, if variable j has been zero in a_{i_k} for 5 iterations, then we can compute the needed value x_j^k using

$$x_j^k = x_j^{k-5} - \frac{5\alpha}{n}\sum_{i=1}^{n}(y_i^k)_j.$$

This update allows SAG to be efficiently applied to sparse data sets where n and p are both in the millions or higher but the number of non-zeros is much less than np.

4.2 Re-weighting on early iterations

In the update of x in Algorithm 1, we normalize the direction d by the total number of data points n. When initializing with $y_i^0 = 0$ we believe this leads to steps that are too small on early iterations of the algorithm where we have only seen a fraction of the data points, because many y_i variables contributing to d are set to the uninformative zero-vector. Following Blatt et al. [5], the more logical normalization is to divide d by m, the number of data points that we have seen at least once (which converges to n once we have seen the entire data set), leading to the update $x = x - \frac{\alpha}{m}d$. Although this modified SAG method appears more difficult to analyze, in our experiments we found that running the basic SAG algorithm from the very beginning with this modification outperformed the basic SAG algorithm as well as the SG/SAG hybrid algorithm mentioned in the Sect. 3. In addition to using the gradient information collected during the first k iterations, this modified SAG algorithm is also advantageous over hybrid SG/SAG algorithms because it only requires estimating a single constant step size.

4.3 Exact and efficient regularization

In the case of regularized objectives like (2), the cost of computing the gradient of the regularizer is independent of n. Thus, we can use the exact gradient of the regularizer in the update of x, and only use d to approximate the sum of the l_i' functions. By incorporating the gradient of the regularizer explicitly, the update for y_i in Algorithm 1 becomes $y_i = l_i'(x)$, and in the case of ℓ_2-regularization the update for x becomes

$$x = x - \alpha\left(\frac{1}{m}d + \lambda x\right) = (1 - \alpha\lambda)\, x - \frac{\alpha}{m}d.$$

If the loss function gradients l_i' are sparse as in Sect. 4.1, then these modifications lead to a reduced storage requirement even though the gradient of the regularizer is dense. Further, although the update of x again appears to require dense vector operations, we can implement the algorithm efficiently if the a_i are sparse. In particular, to allow efficient multiplication of x by the scalar $(1-\alpha\lambda)$, it is useful to represent x in the form $x = \kappa z$, where κ is a scalar and z is a vector (as done by Shalev-Shwartz et al. [55]). Under this representation, we can multiply x by a scalar in $O(1)$ by simply updating κ (though to prevent κ becoming too large or too small we may need to occasionally re-normalize by setting $z = \kappa z$ and $\kappa = 1$). To efficiently implement the vector subtraction operation, we can use a variant of the just-in-time updates from Sect. 4.1. In Algorithm 2, we give pseudo-code for a variant of SAG that includes all of these modifications, and thus uses no full-vector operations. This code uses a vector y to keep track of the scalars $l_i'(u_i^k)$, a vector C to determine whether a data point has previously been visited, a vector V to track the last time a variable was updated, and a vector S to keep track of the cumulative sums needed to implement the just-in-time updates.

4.4 Warm starting

In many scenarios we may need to solve a set of closely-related optimization problems. For example, we may want to apply Algorithm 2 to a regularized objective of the form (2) for several values of the regularization parameter λ. Rather than solving these problems independently, we might expect to obtain better performance by warm-starting the algorithm. Although initializing x with the solution of a related problem can improve performance, we can expect an even larger performance improvement if we also use the gradient information collected from a run of SAG for a close value of λ. For example, in Algorithm 2 we could initialize x, y_i, d, m, and C_i based on a previous run of the SAG algorithm. In this scenario, Theorem 1 suggests that it may be beneficial in this setting to center the y_i variables around d.

4.5 Larger step sizes

In our experiments we have observed that utilizing a step size of $1/L$, as in standard FG methods, always converged and often performed better than the step size of $1/16L$ suggested by our analysis. Thus, in our experiments we used $\alpha_k = 1/L$ even though

Algorithm 2 SAG variant for minimizing $\frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} l_i(a_i^\top x)$, with step size α and a_i sparse.

{Initialization, note that $x = \kappa z$.}
$d = 0$, $y_i = 0$ for $i = 1, 2, \ldots, n$
$z = x$, $\kappa = 1$
$m = 0$, $C_i = 0$ for $i = 1, 2, \ldots, n$
$S_{-1} = 0$, $V_j = 0$ for $j = 1, 2, \ldots, p$
for $k = 0, 1, \ldots$ **do**
  Sample i from $\{1, 2, \ldots, n\}$
  if $C_i = 0$ **then**
    {This is the first time we have sampled this data point.}
    $m = m + 1$
    $C_i = 1$
  end if
  for j non-zero in a_i **do**
    {Just-in-time calculation of needed values of z.}
    $z_j = z_j - (S_{k-1} - S_{V_j - 1})d_j$
    $V_j = k$
  end for
  {Update the memory y and the direction d.}
  Let J be the support of a_i
  $d_J = d_J - a_{iJ}(y_i - l_i'(\kappa a_{iJ}^T z_J))$
  $y_i = l_i'(\kappa a_{iJ}^T z_J)$
  {Update κ and the sum needed for z updates.}
  $\kappa = \kappa(1 - \alpha\lambda)$
  $S_k = S_{k-1} + \alpha/(\kappa m)$
end for
{Final x is κ times the just-in-time update of all z.}
for $j = 1, 2, \ldots, p$ **do**
  $x_j = \kappa(z_j - (S_{k-1} - S_{V_j - 1})d_j)$
end for

we do not have a formal analysis of the method under this step size. We also found that a step size of $2/(L + n\mu)$, which in the strongly-convex case corresponds to the best fixed step size for the FG method in the special case of $n = 1$ [42, see Theorem 2.1.15], sometimes yields even better performance (though in other cases it performs poorly).

4.6 Line-search when L is not known

In general the Lipschitz constant L will not be known, but we may obtain a reasonable approximation of a valid L by evaluating f_i values while running the algorithm. In our experiments, we used a basic line-search where we start with an initial estimate L^0, and double this estimate whenever we do not satisfy the inequality

$$f_{i_k}\left(x^k - \frac{1}{L^k} f_{i_k}'(x^k)\right) \leqslant f_{i_k}(x^k) - \frac{1}{2L^k}\|f_{i_k}'(x^k)\|^2,$$

which must be true if L^k is valid. An important property of this test is that it depends on f_{i_k} but not on g, and thus the cost of performing this test is independent of n. To avoid instability caused by comparing very small numbers, we only do this test when

$\|f'_{i_k}(x^k)\|^2 > 10^{-8}$. Since L is a global quantity but the algorithm will eventually remain within a neighbourhood of the optimal solution, it is possible that a smaller estimate of L (and thus a larger step size) can be used as we approach x^*. To potentially take advantage of this, we initialize with the slightly smaller $L^k = (L^{k-1}2^{-1/n})$ at each iteration, so that the estimate of L is halved if we do n iterations (an effective pass through the data) and never violate the inequality. Note that in the case of ℓ_2-regularized objectives, we can perform the line-search to find an estimate of the Lipschitz constant of l'_i rather than f'_i, and then simply add λ to this value to take into account the effect of the regularizer.

Note that the cost of this line-search is *independent* of n, making it suitable for large problems. Further, for linearly-parameterized models of the form $f_i(a_i^T x)$, it is also possible to implement the line-search so that its cost is also independent of the number of variables p. To see why, if we use $\delta^k = a_{i_k}^T x^k$ and the structure in the gradient then the left side is given by

$$f_{i_k}\left(a_{i_k}^T\left(x^k - \frac{1}{L^k} f'_{i_k}(x^k)\right)\right) = f_{i_k}\left(\delta^k - \frac{f'_{i_k}(\delta^k)}{L^k}\|a_{i_k}\|^2\right).$$

Thus, if we pre-compute the squared norms $\|a_i\|^2$ and note that δ^k and $f'_{i_k}(\delta^k)$ are already needed by the SAG update, then each iteration only involves operations on scalar values and the single-variable function f_{i_k}.

4.7 Mini-batches for vectorized computation and reduced storage

Because of the use of vectorization and parallelism in modern architectures, practical SG implementations often group functions into 'mini-batches' and perform SG iterations on the mini-batches. We can also use mini-batches within the SAG iterations to take advantage of the same vectorization and parallelism. Additionally, for problems with dense gradients mini-batches can dramatically decrease the storage requirements of the algorithm, since we only need to store a vector y_i for each mini-batch rather than for each example. Thus, for example, using a mini-batch of size 100 leads to a 100-fold reduction in the storage cost.

A subtle issue that arises when using mini-batches is that the value of L in the Lipschitz condition (8) is based on the mini-batches instead of the original functions f_i. For example, consider the case where we have a batch $\mathcal{B}$ and the minimum value of L in (8) for each i is given by L_i. In this case, a valid value of L for the function $x \mapsto \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} f_i(x)$ would be $\max_{i\in\mathcal{B}}\{L_i\}$. We refer to this as $L_{\max}$. But we could also consider using $L_{\text{mean}} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} L_i$. The value L_{mean} is still valid and will be smaller than $L_{\max}$ unless all L_i are equal. We could even consider the minimum possible value of L, which we refer to as L_{Hessian} because (if each f_i is twice-differentiable) it is equal to the maximum eigenvalue of $\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} f''_i(x)$ across all x. Note that $L_{\text{Hessian}} \leq L_{\text{mean}} \leq L_{\max}$, although L_{Hessian} will typically be more difficult to compute than L_{mean} or $L_{\max}$ (although a line-search as discussed in the previous section can reduce this cost). Due to the potential of using a smaller L, *we may obtain a faster*

convergence rate by using larger mini-batches. However, in terms of passes through the data this faster convergence may be offset by the higher iteration cost associated with using mini-batches.

4.8 Non-uniform example selection

In standard SG methods, it is crucial to sample the functions f_i uniformly, at least asymptotically, in order to yield an unbiased gradient estimate and subsequently achieve convergence to the optimal value (alternately, the bias induced by non-uniform sampling would need to be asymptotically corrected). In SAG iterations, however, the weight of each gradient is constant and equal to $1/n$, regardless of the frequency at which the corresponding function is sampled. We might thus consider sampling the functions f_i non-uniformly, without needing to correct for this bias. Though we do not yet have any theoretical proof as to why a non-uniform sampling might be beneficial, intuitively we would expect that we do not need to sample functions f_i whose gradient changes slowly as often as functions f_i whose gradient changes more quickly. Indeed, we provide here an argument to justify a non-uniform sampling strategy based on the Lipschitz constants of the individual gradients f_i' and we note that in subsequent works this intuition has proved correct for related algorithms [53,64].

Let L_i again be the Lipschitz constant of f_i', and assume that the functions are placed in increasing order of Lipschitz constants, so that $L_1 \leqslant L_2 \leqslant \ldots \leqslant L_n$. In the ill-conditioned setting where the convergence rate depends on $\frac{\mu}{L}$, a simple way to improve the rate by decreasing L is to replace f_n by two functions f_{n1} and f_{n2} such that

$$
\begin{aligned}
f_{n1}(x) = f_{n2}(x) &= \frac{f_n(x)}{2} \\
g(x) &= \frac{1}{n}\left(\sum_{i=1}^{n-1} f_i(x) + f_{n1}(x) + f_{n2}(x)\right) \\
&= \frac{1}{n+1}\left(\sum_{i=1}^{n-1} \frac{n+1}{n} f_i(x) + \frac{n+1}{n} f_{n1}(x) + \frac{n+1}{n} f_{n2}(x)\right).
\end{aligned}
$$

We have thus replaced the original problem by a new, equivalent problem where:

- n has been replaced by $(n+1)$,
- L_i for $i \leqslant (n-1)$ is $\frac{L_i(n+1)}{n}$,
- L_n and L_{n+1} are equal to $\frac{L_n(n+1)}{2n}$.

Hence, if $L_{n-1} < \frac{nL_n}{n+1}$, this problem has the same μ but a smaller L than the original one, improving the bound on the convergence rate. By duplicating f_n, we increase its probability of being sampled from $\frac{1}{n}$ to $\frac{2}{n+1}$, but we also replace y_n^k by a noisier version, i.e. $y_{n1}^k + y_{n2}^k$. Using a noisier version of the gradient appears detrimental, so we assume that the improvement comes from increasing the frequency at which f_n is sampled, and that logically we might obtain a better rate by simply sampling f_n more often in the original problem and not explicitly duplicating the data.

We now consider the extreme case of duplicating each function f_i a number of times equal to the Lipschitz constant of their gradient, assuming that these constants are integers. The new problem becomes

$$\begin{aligned} g(x) &= \frac{1}{n}\sum_{i=1}^{n} f_i(x) \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L_i}\frac{f_i(x)}{L_i} \\ &= \frac{1}{\sum_k L_k}\sum_{i=1}^{n}\sum_{j=1}^{L_i}\left(\frac{\sum_k L_k}{n}\frac{f_i(x)}{L_i}\right). \end{aligned}$$

The function g is now written as the sum of $\sum_k L_k$ functions, each with a gradient with Lipschitz constant $\frac{\sum_k L_k}{n}$. The new problem has the same μ as before, but now has an L equal to the average of the Lipschitz constants across the f_i', rather than their maximum, thus improving the bound on the convergence rate. Sampling these functions uniformly is now equivalent to sampling the original f_i's according to their Lipschitz constant. Thus, we might expect to obtain better performance by, instead of creating a larger problem by duplicating the functions in proportion to their Lipschitz constant, simply sampling the functions from the original problem in proportion to their Lipschitz constants.

Sampling in proportion to the Lipschitz constants of the gradients was explored by Nesterov [45] in the context of coordinate descent methods, and is also somewhat related to the sampling scheme used by Storhmer and Vershynin [59] in the context of their randomized Kaczmarz algorithm. Since the first version of this work was released, Needell et al. [38] have analyzed sampling according to the Lipschitz constant in the context of SG iterations. Such a sampling scheme makes the iteration cost depend on n, due to the need to generate samples from a general discrete distribution over n variables. However, after an initial preprocessing cost of $O(n)$ we can sample from such distributions in $O(\log n)$ using a simple binary search [50, see Example 2.10].

Unfortunately, sampling the functions according to the Lipschitz constants and using a step size of $\alpha_k = \frac{n}{\sum_i L_i}$ does not seem to converge in general. However, by changing the number of times we duplicate each f_i, we can interpolate between the Lipschitz sampling and the uniform sampling. In particular, if each function f_i is duplicated $L_i + c$ times, where L_i is the Lipschitz constant of f_i' and c a positive number, then the new problem becomes

$$\begin{aligned} g(x) &= \frac{1}{n}\sum_{i=1}^{n} f_i(x) \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L_i+c}\frac{f_i(x)}{L_i+c} \end{aligned}$$

$$= \frac{1}{\sum_k (L_k + c)} \sum_{i=1}^{n} \sum_{j=1}^{L_i + c} \left(\frac{\sum_k (L_k + c)}{n} \frac{f_i(x)}{L_i + c} \right).$$

Unlike in the previous case, these $\sum_k (L_k + c)$ functions have gradients with different Lipschitz constants. Denoting $L = \max_i L_i$, the maximum Lipschitz constant is equal to $\frac{\sum_k (L_k + c)}{n} \frac{L}{L+c}$ and we must thus use the step size $\alpha = \frac{L+c}{L\left(\frac{\sum_k L_k}{n} + c\right)}$.

5 Experimental results

In this section we perform empirical evaluations of the SAG iterations. We first compare the convergence of an implementation of the SAG iterations to a variety of competing methods available. We then seek to evaluate the effect of different of non-uniform sampling. In Sect. 5 of the extended version of this paper, we present further experiments evaluating the effect of the step-size and the effect of using mini-batches.

5.1 Comparison to FG and SG methods

The theoretical convergence rates suggest the following strategies for deciding on whether to use an FG or an SG method:

- If we can only afford one pass through the data, then an SG method should be used.
- If we can afford to do many passes through the data (say, several hundred), then an FG method should be used.

We expect that the SAG iterations will be most useful between these two extremes, where we can afford to do more than one pass through the data but cannot afford to do enough passes to warrant using FG algorithms like the AFG or L-BFGS methods. To test whether this is indeed the case in practice, we perform a variety of experiments evaluating the performance of the SAG algorithm in this scenario.

Although the SAG algorithm can be applied more generally, in our experiments we focus on the important and widely-used ℓ_2-regularized logistic regression problem

$$\underset{x \in \mathbb{R}^p}{\text{minimize}} \quad \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-b_i a_i^\top x)), \tag{10}$$

as a canonical problem satisfying our assumptions. In our experiments we set the regularization parameter λ to $1/n$, which is in the range of the smallest values that would typically be used in practice, and thus which results in the most ill-conditioned problems of this form that would be encountered. Our experiments focus on the freely-available benchmark binary classification data sets listed in Table 2. The *quantum* and *protein* data set was obtained from the KDD Cup 2004 website;[1] the *covertype* (based

[1] http://osmot.cs.cornell.edu/kddcup.

Table 2 Binary data sets used in the experiments

Data set	Data points	Variables	References
quantum	50000	78	[9]
protein	145751	74	[9]
covertype	581012	54	[16]
rcv1	20242	47236	[31]
news	19996	1355191	[24]
spam	92189	823470	[8,12]
rcv1Full	697641	47236	[31]
sido	12678	4932	[20]
alpha	500000	500	Synthetic

on the datset of Blackard, Jock, and Dean), *rcv1*, *news*, and *rcv1Full* data sets were obtained from the LIBSVM Data website;[2] the *sido* data set was obtained from the Causality Workbench website,[3] the *spam* data set was prepared by Carbonetto [8, see §2.6.5] using the TREC 2005 corpus;[4] and the *alpha* data set was obtained from the Pascal Large Scale Learning Challenge website.[5] We added a (regularized) bias term to all data sets, and for dense features we standardized so that they would have a mean of zero and a variance of one. To obtain results that are independent of the practical implementation of the algorithm, we measure the objective as a function of the number of effective passes through the data, measured as the number of times we evaluate l_i' divided by the number of examples n. If they are implemented to take advantage of the sparsity present in the data sets, the runtimes of all algorithms examined in this section differ by at most a constant times this measure.

In our first experiment we compared the following variety of competitive FG and SG methods:

- *AFG* A variant of the accelerated full gradient method of Nesterov [41], where iterations of (3) with a step size of $1/L^k$ are interleaved with an extrapolation step. We used an adaptive line-search to estimate a local L based on the variant proposed for ℓ_2-regularized logistic regression by Liu et al. [33].
- *L-BFGS* A publicly-available limited-memory quasi-Newton method that has been tuned for log-linear models such as logistic regression [51]. This method is the most complicated method we considered.
- *SG* The stochastic gradient method described by iteration (4). Since setting the step-size in this method is a tenuous issue, we chose the constant step size that gave the best performance (in hindsight) among all powers of 10 (we found that this constant step-size strategies gave better performance than the variety of decreasing step-size strategies that we experimented with).

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

[3] http://www.causality.inf.ethz.ch/home.php.

[4] http://plg.uwaterloo.ca/~gvcormac/treccorpus.

[5] http://largescale.ml.tu-berlin.de.

- *ASG* The average of the iterations generated by the SG method above, where again we choose the best step size among all powers of 10.[6]
- *IAG* The incremental aggregated gradient method of Blatt et al. [5] described by iteration (5) with a cyclic choice of i_k. We used the re-weighting described in Sect. 4.2, we used the exact regularization as described in Sect. 4.3, and we chose the step-size that gave the best performance among all powers of 10.
- *SAG-LS* The proposed stochastic average gradient method described by iteration (5). We used the re-weighting described in Sect. 4.2, the exact regularization as described in Sect. 4.3, and we used a step size of $\alpha_k = 1/L^k$ where L^k is an approximation of the Lipschitz constant for the negative log-likelihoods $l_i(x) = \log(1 + \exp(-b_i a_i^\top x))$. Although this Lipschitz constant is given by $0.25 \max_i\{\|a_i\|^2\}$, we used the line-search described in Sect. 4.6 to estimate it, to test the ability to use SAG as a black-box algorithm (in addition to avoiding this calculation and potentially obtaining a faster convergence rate by obtaining an estimate that could be smaller than the global value). To initialize the line-search we set $L^0 = 1$.

We plot the results of the different methods for the first 50 effective passes through the data in Fig. 1. For the stochastic methods, we plot the mean performance as well as the minimum and maximum function values across 10 choices for the initial random seed. We can observe several trends across the experiments:

- *FG versus SG* Although the performance of SG methods is known to be catastrophic if the step size is not chosen carefully, after giving the SG methods (*SG* and *ASG*) an unfair advantage (by allowing them to choose the best step-size in hindsight), the SG methods always do substantially better than the FG methods (*AFG* and *L-BFGS*) on the first few passes through the data. However, the SG methods typically make little progress after the first few passes. In contrast, the FG methods make steady progress and eventually the faster FG method (*L-BFGS*) typically passes the SG methods.
- *(FG and SG) versus SAG* The SAG iterations seem to achieve the best of both worlds. They start out substantially better than FG methods, often obtaining similar performance to an SG method with the best step-size chosen in hindsight. But the SAG iterations continue to make steady progress even after the first few passes through the data. This leads to better performance than SG methods on later iterations, and on most data sets the sophisticated FG methods do not catch up to the SAG method even after 50 passes through the data.

[6] Note that we also compared to a variety of other SG methods including the popular Pegasos SG method of Shalev-Shwartz et al. [55], SG with momentum, SG with gradient averaging, the regularized dual averaging method of Xiao [63] (a stochastic variant of the primal-dual subgradient method of Nesterov [44] for regularized objectives), the accelerated SG method of Delyon and Juditsky [14], SG methods that only average the later iterations as in the 'optimal' SG method for non-smooth optimization of Rakhlin et al. [48], the epoch SG method of Hazan and Kale [21], the 'nearly-optimal' SG method of Ghadimi and Lan [18], a diagonally-scaled SG method using the inverse of the coordinate-wise Lipshitz constants as the diagonal terms, and the adaptive diagonally-scaled AdaGrad method of Duchi et al. [15]. However, we omit results obtained using these algorithms since they never performed substantially better than the minimum between the *SG* and *ASG* methods when their step-size was chosen in hindsight.

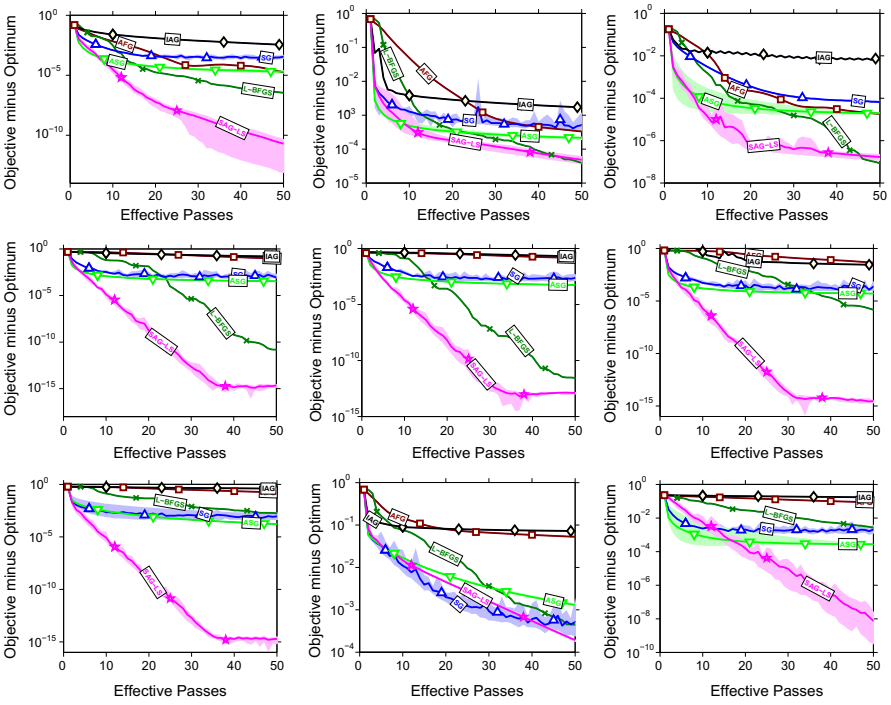

Fig. 1 Comparison of different FG and SG optimization strategies. The *top row* gives results on the *quantum* (*left*), *protein* (*center*) and *covertype* (*right*) datasets. The *middle row* gives results on the *rcv1* (*left*), *news* (*center*) and *spam* (*right*) datasets. The *bottom row* gives results on the *rcv1Full* (*left*), *sido* (*center*), and *alpha* (*right*) datasets. This figure is best viewed in *colour*

– *IAG versus SAG* Even though these two algorithms differ in only the seemingly-minor detail of selecting data points at random (SAG) compared to cycling through the data (IAG), the performance improvement of SAG over its deterministic counterpart IAG is striking (even though the IAG method was allowed to choose the best step-size in hindsight). We believe this is due to the larger step sizes allowed by the SAG iterations, which would cause the IAG iterations to diverge.

5.2 Comparison to coordinate optimization methods

For the ℓ_2-regularized logistic regression problem, an alternative means to take advantage of the structure of the problem and achieve a linear convergence rate with a cheaper iteration cost than FG methods is to use randomized coordinate optimization methods. In particular, we can achieve a linear convergence rate by applying coordinate descent to the primal [45] or coordinate-ascent to the dual [54]. In our second experiment, we included the following additional methods in this comparison:

– *PCD* The randomized primal coordinate-descent method of Nesterov [45], using a step-size of $1/L_j$, where L_j is the Lipschitz-constant with respect to coordinate j of g'. Here, we sampled the coordinates uniformly.

– *PCD-L* The same as above, but sampling coordinates according to their Lipschitz constant, which can lead to an improved convergence rate [45].
– *DCA* Applying randomized coordinate ascent to the dual, with uniform example selection and an exact line-search [54].

As with comparing FG and SG methods, it is difficult to compare coordinate-wise methods to FG and SG methods in an implementation-independent way. To do this in a way that we believe is fair (when discussing convergence rates), we measure the number of effective passes of the *DCA* method as the number of iterations of the method divided by n (since each iteration accesses a single example as in SG and SAG iterations). We measure the number of effective passes for the *PCD* and *PCD-L* methods as the number of iterations multiplied by n/p so that 1 effective pass for this method has a cost of $O(np)$ as in FG and SG methods. We ignore the additional cost associated with the Lipschitz sampling for the *PCD-L* method (as well as the expense incurred because the *PCD-L* method tended to favour updating the bias variable for sparse data sets) and we also ignore the cost of numerically computing the optimal step-size for the *DCA* method.

We compare the performance of the randomized coordinate optimization methods to several of the best methods from the previous experiment in Fig. 2. In these experiments we observe the following trends:

– *PCD versus PCD-L* For the problems with $n > p$ (top and bottom rows of Fig. 2), there is little difference between uniform and Lipschitz sampling of the coordinates. For the problems with $p > n$ (middle row of Fig. 2), sampling according to the Lipschitz constant leads to a large performance improvement over uniform sampling.
– *PCD versus DCA* For the problems with $p > n$, *DCA* and *PCD-L* have similar performance. For the problems with $n > p$, the performance of the methods typically differed but neither strategy tended to dominate the other.
– *(PCD and DCA) versus (SAG)* For some problems, the *PCD* and *DCA* methods have performance that is similar to *SAG-LS* and the *DCA* method even gives better performance than *SAG-LS* on one data set. However, for many data sets either the *PCD-L* or the *DCA* method (or both) perform poorly while the *SAG-LS* iterations are among the best or substantially better than all other methods on every data set. This suggests that, while coordinate optimization methods are clearly extremely effective for some problems, the SAG method tends to be a more robust choice across problems.

5.3 Effect of non-uniform sampling

In our final experiment, we explored the effect of using the non-uniform sampling strategy discussed in Sect. 4.8. In Fig. 3, we compare several of the SAG variants with uniform sampling to the following two methods based on non-uniform sampling:

– *SAG (Lipschitz)* In this method we sample the functions in proportion to $L_i + c$, where L_i is the global Lipschitz constant of the corresponding f_i' and we set c to the average of these constants, $c = L_{\text{mean}} = (1/n)\sum_i L_i$. Plugging in these values

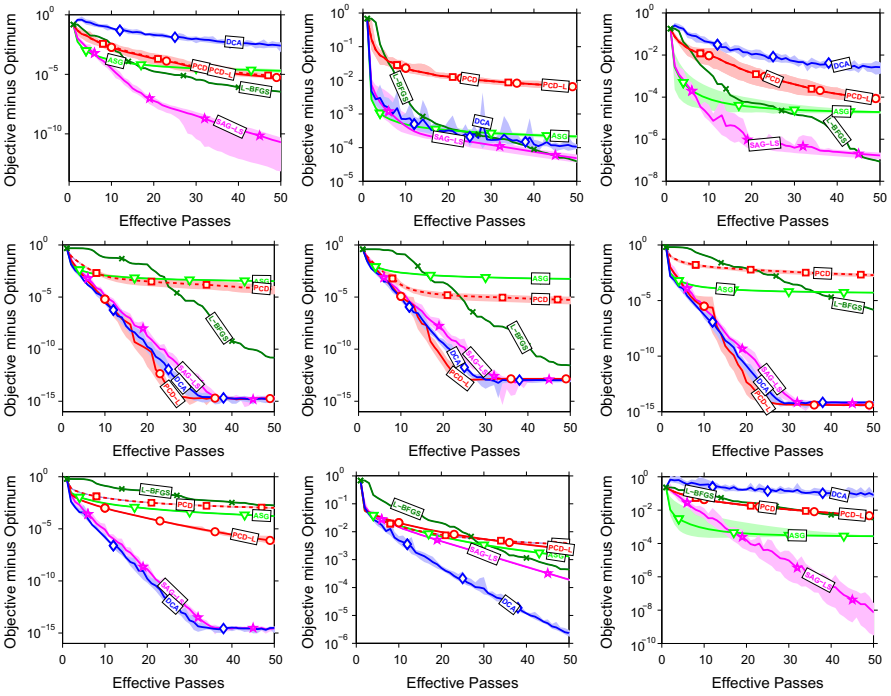

Fig. 2 Comparison of optimization different FG and SG methods to coordinate optimization methods. The *top row* gives results on the *quantum* (*left*), *protein* (*center*) and *covertype* (*right*) datasets. The *middle row* gives results on the *rcv1* (*left*), *news* (*center*) and *spam* (*right*) datasets. The *bottom row* gives results on the *rcv1Full* (*left*), *sido* (*center*), and *alpha* (*right*) datasets. This figure is best viewed in *colour*

into the formula at the end of Sect. 4.8 and using $L_{\max}$ to denote the maximum L_i value, we set the step-size to $\alpha_k = 1/2L_{\max} + 1/2L_{\text{mean}}$.

– *SAG-LS (Lipschitz)* In this method we formed estimates of the quantities L_i, $L_{\max}$, and L_{mean}. The estimator $L^k_{\max}$ is computed in the same way as the SAG-LS method. To estimate each L_i, we keep track of an estimate L^k_i for each i and we set L^k_{mean} to the average of the L^k_i values among the f_i that we have sampled at least once. We set $L^k_i = L^{k-1}_i$ if example i was not selected and otherwise we initialize to $L^k_i = L^{k-1}_i/2$ and perform the line-search until we have a valid L^k_i (this means that L^k_{mean} will be approximately halved if we perform a full pass through the data set and never violate the inequality). To ensure that we do not ignore important unseen data points for too long, in this method we sample a previously unseen function with probability $(n - m)/n$, and otherwise we sample from the previously seen f_i in proportion to $L^k_i + L^k_{\text{mean}}$. To prevent relying too much on our initially-poor estimate of L_{mean}, we use a step size of $\alpha_k = \frac{n-m}{n}\alpha_{\max} + \frac{m}{n}\alpha_{\text{mean}}$, where $\alpha_{\max} = 1/L^k_{\max}$ is the step-size we normally use with uniform sampling and $\alpha_{\text{mean}} = 1/2L^k_{\max} + 1/2L^k_{\text{mean}}$ is the step-size we use with the non-uniform sampling method, so that the method interpolates between these extremes until the entire data set has been sampled.

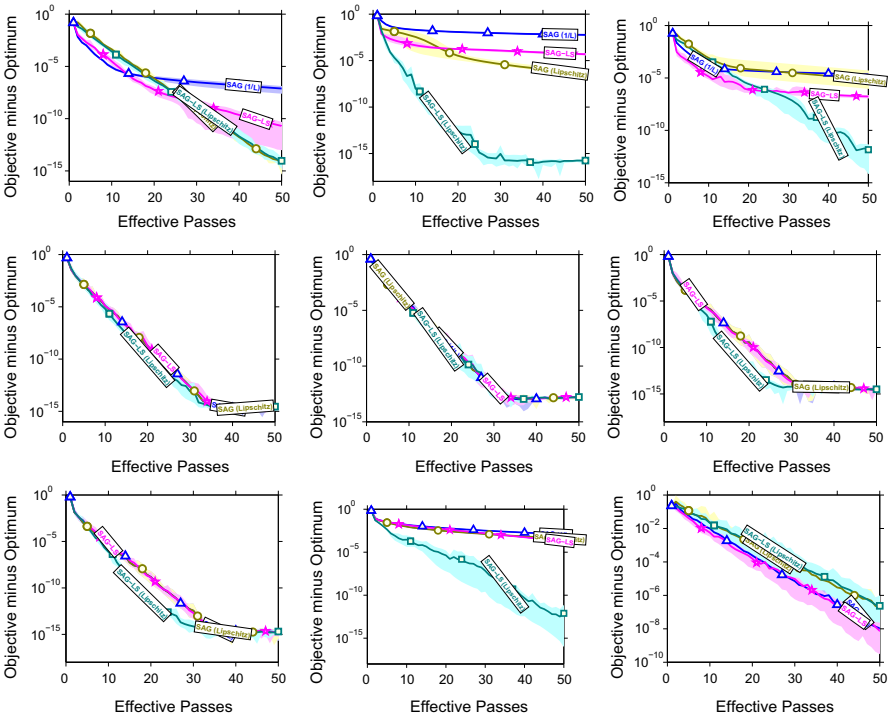

Fig. 3 Comparison of uniform and non-uniform sampling strategies for the SAG algorithm. The *top row* gives results on the *quantum* (*left*), *protein* (*center*) and *covertype* (*right*) datasets. The *middle row* gives results on the *rcv1* (*left*), *news* (*center*) and *spam* (*right*) datasets. The *bottom row* gives results on the *rcv1Full* (*left*), *sido* (*center*), and *alpha* (*right*) datasets. This figure is best viewed in *colour*

We make the following observations from these experiments:

- *SAG (1/L) versus SAG (Lipschitz)* With access to global quantities and a constant step size, the difference between uniform and non-uniform sampling was typically quite small. However, in some cases the non-uniform sampling method behaved much better (top row of Fig. 3).
- *SAG-LS versus SAG-LS (Lipschitz)* When estimating the Lipschitz constants of the individual functions, the non-uniform sampling strategy often gave better performance. Indeed, the adaptive non-uniform sampling strategy gave solutions that are orders of magnitude more accurate than any method we examined for several of the data sets (e.g., the *protein*, *covertype*, and *sido* data sets) In the context of logistic regression, it makes sense that an adaptive sampling scheme could lead to better performance, as many correctly-classified data samples might have a very slowly-changing gradient near the solution, and thus they do not need to be sampled often.

6 Discussion

Since the first version of this work was published [30], there has been an explosion of interest in stochastic methods with improved convergence rates. In Sect. 6 of the extended version of this paper we review other algorithms that have been discovered to have similar convergence rates, including stochastic dual coordinate ascent (SDCA) [54], incremental surrogate optimization (MISO) [35], stochastic variance-reduced gradient (SVRG) methods [23,26,34] which remove the memory requirement, and the SAGA method [13]. In Sect. 6 of the extended version we also review many of the possible variants on these basic algorithms that have been explored. This includes accelerated gradient methods that achieve faster convergence rates for ill-conditioned problems [32], proximal-gradient and alternating direction method of multipliers (ADMM) variants that can solve certain constrained and non-smooth problems [13,65], coordinate-wise variants that only update a subset of the variables on each iteration [27], Newton-like variants of the method [56], methods where non-uniform sampling provably improves the convergence rate [53], and analyses that give a linear convergence rate without strong convexity [19].

The three major disadvantages of SG methods are: (i) the slow convergence rate, (ii) deciding when to terminate the algorithms, and (iii) choosing the step size while running the algorithm. This work shows that the SAG iterations achieve a much faster convergence rate, but the SAG iterations may also be advantageous in terms of termination criteria and choosing step sizes. In particular, the SAG iterations suggest a natural termination criterion; since the quantity d in Algorithm 1 converges to $f'(x^k)$ as $\|x^k - x^{k-1}\|$ converges to zero, we can use $\|d\|$ as an approximation of the optimality of x^k as the iterates converge. Regarding choosing the step-size, a disadvantage of a constant step-size strategy is that a step-size that is too large may cause divergence. But, we expect that it is possible to design line-search or trust-region strategies that avoid this issue. Such strategies might even lead to faster convergence for functions that are locally well-behaved around their optimum, as indicated in our experiments. Further, while SG methods require specifying a sequence of step sizes and mis-specifying this sequence can have a disastrous effect on the convergence rate [40, see §2.1], our theory shows that the SAG iterations achieve a fast convergence rate for any sufficiently small constant step size, and our experiments indicate that a simple line-search gives strong performance.

Acknowledgements We would like to thank the anonymous reviewers for their many useful comments. This work was partially supported by the European Research Council (SIERRA-ERC-239993) and a Google Research Award. Mark Schmidt is also supported by a postdoctoral fellowship from the Natural Sciences and Engineering Research Council of Canada.

References

1. Agarwal, A., Bartlett, P.L., Ravikumar, P., Wainwright, M.J.: Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization. IEEE Trans. Inf. Theory **58**(5), 3235–3249 (2012)
2. Bach, F., Moulines, E.: Non-asymptotic analysis of stochastic approximation algorithms for machine learning. Adv. Neural Inf. Process. Syst. 773–781 (2013)

3. Bach, F., Moulines, E.: Non-strongly-convex smooth stochastic approximation with convergence rate $o(1/n)$. arXiv preprint (2013)
4. Bertsekas, D.P.: A new class of incremental gradient methods for least squares problems. SIAM J. Optim. **7**(4), 913–926 (1997)
5. Blatt, D., Hero, A.O., Gauchman, H.: A convergent incremental gradient method with a constant step size. SIAM J. Optim. **18**(1), 29–51 (2007)
6. Bordes, A., Bottou, L., Gallinari, P.: Sgd-qn: careful quasi-newton stochastic gradient descent. J. Mach. Learn. Res. **10**, 1737–1754 (2009)
7. Bottou, L., LeCun, Y.: Large scale online learning. Adv. Neural Inf. Process. Syst. 217–224 (2003)
8. Carbonetto, P.: New probabilistic inference algorithms that harness the strengths of variational and Monte Carlo methods. Ph.D. thesis, University of British Columbia (2009)
9. Caruana, R., Joachims, T., Backstrom, L.: KDD-cup 2004: results and analysis. ACM SIGKDD Newsl. **6**(2), 95–108 (2004)
10. Cauchy, A.: Méthode générale pour la résolution des systèmes d'équations simultanées. Comptes rendus des séances de l'Académie des sciences de Paris **25**, 536–538 (1847)
11. Collins, M., Globerson, A., Koo, T., Carreras, X., Bartlett, P.: Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. J. Mach. Learn. Res. **9**, 1775–1822 (2008)
12. Cormack, G.V., Lynam, T.R.: Spam corpus creation for TREC. In: Proceedings of 2nd Conference on Email and Anti-Spam (2005). http://plg.uwaterloo.ca/~gvcormac/treccorpus/
13. Defazio, A., Bach, F., Lacoste-Julien, S.: Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. Adv. Neural Inf. Process. Syst. 1646–1654 (2014)
14. Delyon, B., Juditsky, A.: Accelerated stochastic approximation. SIAM J. Optim. **3**(4), 868–881 (1993)
15. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
16. Frank, A., Asuncion, A.: UCI machine learning repository (2010). http://archive.ics.uci.edu/ml
17. Friedlander, M.P., Schmidt, M.: Hybrid deterministic-stochastic methods for data fitting. SIAM J. Sci. Comput. **34**(3), A1351–A1379 (2012)
18. Ghadimi, S., Lan, G.: Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization. Optim. Online (2010)
19. Gong, P., Ye, J.: Linear convergence of variance-reduced projected stochastic gradient without strong convexity. arXiv preprint (2014)
20. Guyon, I.: Sido: A phamacology dataset (2008). http://www.causality.inf.ethz.ch/data/SIDO.html
21. Hazan, E., Kale, S.: Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. J. Mach. Learn. Res. Workshop Conf. Proc. **15**, 2489–2512 (2011)
22. Hu, C., Kwok, J., Pan, W.: Accelerated gradient methods for stochastic optimization and online learning. Adv. Neural Inf. Process. Syst. 781–789 (2009)
23. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. Adv. Neural Inf. Process. Syst. 315–323 (2013)
24. Keerthi, S., DeCoste, D.: A modified finite Newton method for fast solution of large scale linear SVMs. J. Mach. Learn. Res. **6**, 341–361 (2005)
25. Kesten, H.: Accelerated stochastic approximation. Ann. Math. Stat. **29**(1), 41–59 (1958)
26. Konečnỳ, J., Richtárik, P.: Semi-stochastic gradient descent methods. arXiv preprint (2013)
27. Konečnỳ, J., Qu, Z., Richtárik, P.: Semi-stochastic coordinate descent. arXiv preprint (2014)
28. Kushner, H.J., Yin, G.: Stochastic Approximation and Recursive Algorithms and Applications, 2nd edn. Springer, New york (2003)
29. Lacoste-Julien, S., Jaggi, M., Schmidt, M., Pletscher, P.: Block-coordinate frank-wolfe optimization for structural SVMs. Int. Conf. Mach. Learn. J. Mach. Learn. Res. Workshop Conf. Proc. **28**, 53–61 (2013)
30. Le Roux, N., Schmidt, M., Bach, F.: A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. Adv. Neural Inf. Process. Syst. 2663–2671 (2012)
31. Lewis, D., Yang, Y., Rose, T., Li, F.: RCV1: a new benchmark collection for text categorization research. J. Mach. Learn. Res. **5**, 361–397 (2004)
32. Lin, H., Mairal, J., Harchaoui, Z.: A universal catalyst for first-order optimization. Adv. Neural Inf. Process. Syst. 3384–3392 (2015)
33. Liu, J., Chen, J., Ye, J.: Large-scale sparse logistic regression. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2009)

34. Mahdavi, M., Zhang, L., Jin, R.: Mixed Optimization of Smooth Functions. Adv. Neural Inf. Process. Syst. 674–682 (2013)
35. Mairal, J.: Optimization with first-order surrogate functions. J. Mach. Learn. Res. Workshop Conf. Proc. **28**, 783–791 (2013)
36. Martens, J.: Deep learning via Hessian-free optimization. Int. Conf. Mach. Learn. 735–742 (2010)
37. Nedic, A., Bertsekas, D.: Convergence rate of incremental subgradient algorithms. In: Uryasev, S., Pardalos, P.M. (eds.) Stochastic Optimization: Algorithms and Applications, pp. 263–304. Kluwer Academic Publishers, Dordrecht (2000)
38. Needell, D., Srebro, N., Ward, R.: Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. Adv. Neural Inf. Process. Syst. 1017–1025 (2014)
39. Nemirovski, A., Yudin, D.B.: Problem Complexity and Method Efficiency in Optimization. Wiley, New York (1983)
40. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM J. Optim. **19**(4), 1574–1609 (2009)
41. Nesterov, Y.: A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. Doklady AN SSSR **269**(3), 543–547 (1983)
42. Nesterov, Y.: Introductory Lectures on Convex Optimization: A Basic Course. Kluwer Academic Publishers, Dordrecht (2004)
43. Nesterov, Y.: Smooth minimization of non-smooth functions. Math. Program. **103**(1), 127–152 (2005)
44. Nesterov, Y.: Primal-dual subgradient methods for convex problems. Math. Program. **120**(1), 221–259 (2009)
45. Nesterov, Y.: Efficiency of coordinate descent methods on huge-scale optimization problems. CORE Discussion Paper (2010)
46. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, New York (2006)
47. Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM J. Control Optim. **30**(4), 838–855 (1992)
48. Rakhlin, A., Shamir, O., Sridharan, K.: Making gradient descent optimal for strongly convex stochastic optimization. Int. Conf. Mach. Learn. 449–456 (2012)
49. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**(3), 400–407 (1951)
50. Robert, C.P., Casella, G.: Monte Carlo Statistical Methods, 2nd edn. Springer, New York (2004)
51. Schmidt, M.: minfunc: unconstrained differentiable multivariate optimization in matlab. https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html (2005)
52. Schmidt, M., Le Roux, N.: Fast convergence of stochastic gradient descent under a strong growth condition. arXiv preprint (2013)
53. Schmidt, M., Babanezhad, R., Ahemd, M., Clifton, A., Sarkar, A.: Non-uniform stochastic average gradient method for training conditional random fields. Int. Conf. Artif. Intell. Stat. J. Mach. Learn. Res. Workshop Conf. Proc. **38**, 819–828 (2015)
54. Shalev-Schwartz, S., Zhang, T.: Stochastic dual coordinate ascent methods for regularized loss minimization. J. Mach. Learn. Res. **14**, 567–599 (2013)
55. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated sub-gradient solver for SVM. Math. Program. **127**(1), 3–30 (2011)
56. Sohl-Dickstein, J., Poole, B., Ganguli, S.: Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. J. Mach. Learn. Res. Workshop Conf. Proc. **32** (2014)
57. Solodov, M.: Incremental gradient algorithms with stepsizes bounded away from zero. Comput. Optim. Appl. **11**(1), 23–35 (1998)
58. Srebro, N., Sridharan, K.: Theoretical basis for "more data less work"? NIPS Workshop on Computataional Trade-offs in Statistical Learning (2011)
59. Strohmer, T., Vershynin, R.: A randomized Kaczmarz algorithm with exponential convergence. J. Fourier Anal. Appl. **15**(2), 262–278 (2009)
60. Sunehag, P., Trumpf, J., Vishwanathan, S., Schraudolph, N.: Variable metric stochastic approximation theory. J. Mach. Learn. Res. Workshop Conf. Proc. **5**, 560–566 (2009)
61. Teo, C.H., Le, Q., Smola, A.J., Vishwanathan, S.V.N.: A scalable modular convex solver for regularized risk minimization. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2007)
62. Tseng, P.: An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. SIAM J. Optim. **8**(2), 506–531 (1998)
63. Xiao, L.: Dual averaging methods for regularized stochastic learning and online optimization. J. Mach. Learn. Res. **11**, 2543–2596 (2010)

64. Xiao, L., Zhang, T.: A proximal stochastic gradient method with progressive variance reduction. SIAM J. Optim. **24**(2), 2057–2075 (2014)
65. Zhong, L., Kwok, J.: Fast stochastic alternating direction method of multipliers. J. Mach. Learn. Res. Workshop Conf. Proc. **65** (2014)