Minimizing Finite Sums with the Stochastic Average Gradient

Mark Schmidt, Nicolas Le Roux, Francis Bach

Minimizing Finite Sums with the Stochastic Average Gradient

Mark Schmidt
schmidtm@cs.ubc.ca

Nicolas Le Roux
nicolas@le-roux.name

Francis Bach
francis.bach@ens.fr

INRIA - SIERRA Project - Team
Département d'Informatique de l'École Normale Supérieure
Paris, France

January 19, 2015

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

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 [Robbins and Monro, 1951, Bottou and LeCun, 2003]. 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. [2007], and non-smooth loss functions (or regularizers) can also be put in this framework by using smooth approximations (for example, see Nesterov [2005]).

For optimizing problem (1), the standard *deterministic* or *full gradient* (FG) method, which dates back to Cauchy [1847], 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 [see Nesterov, 2004, 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 [see Nesterov, 2004, 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 $\{1,\dots,n\}$. 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 [Nemirovski et al., 2009]) 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 Nemirovski and Yudin [1983], Nemirovski et al. [2009], Agarwal et al. [2012]). 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. [2007]. 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 [2011] 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 Le Roux et al. [2012], 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 (Section 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 (Section 4.8) and present numerical results showing that this can lead to drastically improved performance (Section 5.5).

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.

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 Tseng [1998]. 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

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

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

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

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.

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

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

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 [2009] 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 [Xiao, 2010], it does not improve on the sublinear convergence rates for SG methods.

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 [Polyak and Juditsky, 1992, Bach and Moulines, 2011]. Bather's method [Kushner and Yin, 2003, §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 [Hazan and Kale, 2011, Rakhlin et al., 2012]. However, none of these methods improve on the $O(1/\sqrt{k})$ and $O(1/k)$ rates for SG methods.

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 [1983], 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 [Nocedal and Wright, 2006]). There has been a substantial amount of work on developing stochastic variants of these algorithms, with several of the notable recent examples including Bordes et al. [2009], Hu et al. [2009], Sunehag et al. [2009], Ghadimi and Lan [2010], Martens [2010], Xiao [2010]. Duchi et al. [2011] 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 [Hu et al., 2009, Ghadimi and Lan, 2010, Xiao, 2010]. 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.

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 [2000, 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 [2013], 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) [Polyak and Juditsky, 1992], or under extremely strong assumptions about the relationship between the functions f_i [Solodov, 1998]. 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).

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 [Kesten, 1958, Delyon and Juditsky, 1993]. 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.

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 [1997] 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 [2012] 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.

Incremental Aggregated Gradient: Blatt et al. [2007] 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.

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 [2009] 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 Schmidt and Le Roux [2013]. This is the strong condition required by Solodov [1998] 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.

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 Section 6, 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

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

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 [Le Roux et al., 2012], 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 Le Roux et al. [2012] 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 [2000] 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

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)	$\frac{1}{16L}$	$\left(1-\min\{\frac{\mu}{16L},\frac{1}{8n}\}\right)^n$	0.8825	0.9938

Table 1: Comparison of convergence rates of first-order methods to the convergence rates of n iterations of SAG. In the examples we take $n = 100000$, $L = 100$, $\mu = 0.01$ (Case 1), and $\mu = 0.0001$ (Case 2).

the discussion after Proposition 3.4). Related results also appear in Collins et al. [2008], Lacoste-Julien et al. [2013] 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 [see Nesterov, 2004, §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 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. [2007], 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 Section 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 Section 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. [2011]). 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 Section 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

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

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 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 $n = 1$ [see Nesterov, 2004, 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

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

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 [Xiao and Zhang, 2014, Schmidt et al., 2015].

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 [2010] in the context of coordinate descent methods, and is also somewhat related to the sampling scheme used by Strohmer and Vershynin [2009] in the context of their randomized Kaczmarz algorithm. Since the first version of this work was released, Needell et al. [2014] 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 [see Robert and Casella, 2004, 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

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

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 algorithmic choices such as the step size, mini-batches, and non-uniform sampling.

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

Data set	Data Points	Variables	Reference
quantum	50 000	78	[Caruana et al., 2004]
protein	145 751	74	[Caruana et al., 2004]
covertype	581 012	54	[Frank and Asuncion, 2010]
rcv1	20 242	47 236	[Lewis et al., 2004]
news	19 996	1 355 191	[Keerthi and DeCoste, 2005]
spam	92 189	823 470	[Cormack and Lynam, 2005, Carbonetto, 2009]
rcv1Full	697 641	47 236	[Lewis et al., 2004]
sido	12 678	4 932	[Guyon, 2008]
alpha	500 000	500	Synthetic

Table 2: Binary data sets used in the experiments.

listed in Table 2. The *quantum* and *protein* data set was obtained from the KDD Cup 2004 website;[1] the *covertype* (based 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 [see Carbonetto, 2009, §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 [1983], 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. [2009].

- **L-BFGS**: A publicly-available limited-memory quasi-Newton method that has been tuned for log-linear models such as logistic regression [Schmidt, 2005]. 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).

- **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]

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

[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`

[6] Note that we also compared to a variety of other SG methods including the popular Pegasos SG method of Shalev-Shwartz et al. [2011], SG with momentum, SG with gradient averaging, the regularized dual averaging method of Xiao [2010] (a stochastic variant of the primal-dual subgradient method of Nesterov [2009] for regularized objectives), the

- **IAG**: The incremental aggregated gradient method of Blatt et al. [2007] described by iteration (5) with a cyclic choice of i_k. We used the re-weighting described in Section 4.2, we used the exact regularization as described in Section 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 Section 4.2, the exact regularization as described in Section 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 Section 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 Figure 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 vs. 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) vs. 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.

- **IAG vs. 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

accelerated SG method of Delyon and Juditsky [1993], SG methods that only average the later iterations as in the 'optimal' SG method for non-smooth optimization of Rakhlin et al. [2012], the epoch SG method of Hazan and Kale [2011], the 'nearly-optimal' SG method of Ghadimi and Lan [2010], 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. [2011]. 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.

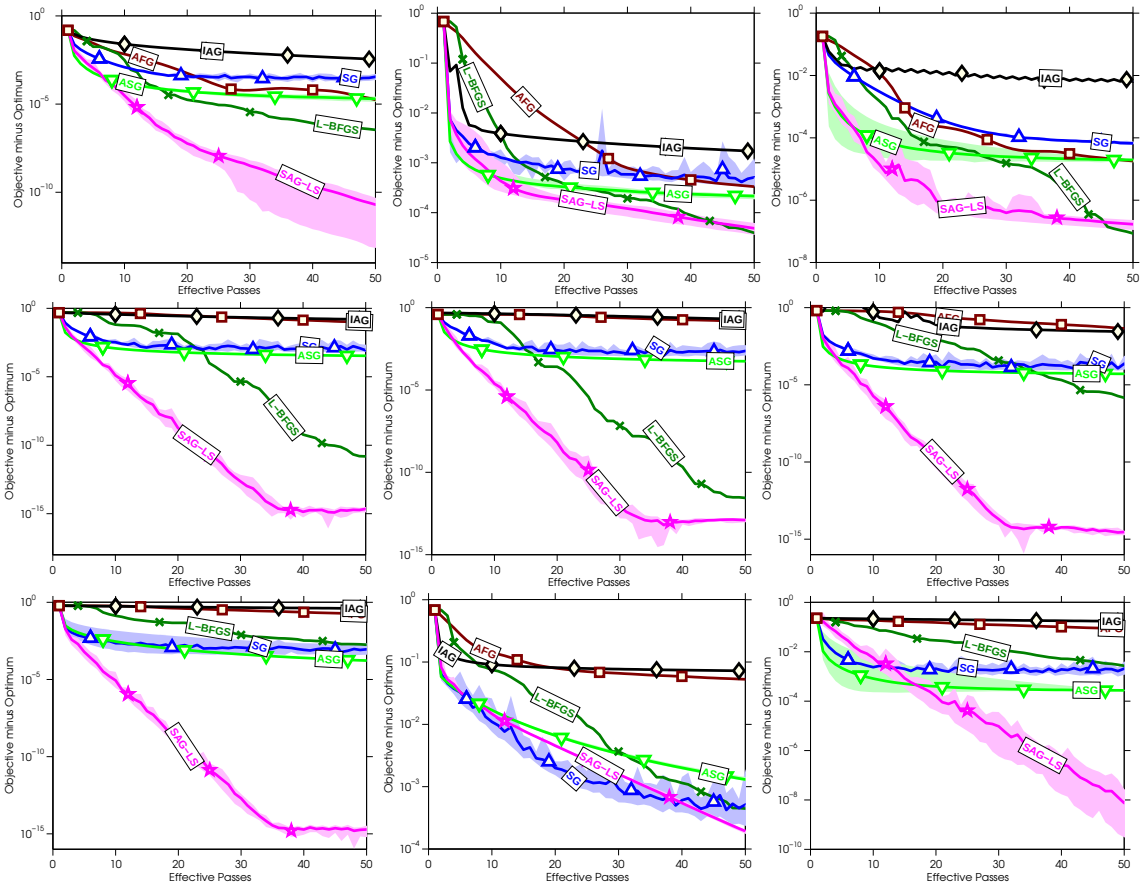

Figure 1: Comparison of different FG and SG optimization strategies. The top row gives results on the *quantum* (left), *protein* (center) and *covertype* (right) datasets. The middle row gives results on the *rcv1* (left), *news* (center) and *spam* (right) datasets. The bottom row gives results on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

linear convergence rate by applying coordinate descent to the primal [Nesterov, 2010] or coordinate-ascent to the dual [Shalev-Schwartz and Zhang, 2013b]. In our second experiment, we included the following additional methods in this comparison:

- **PCD**: The randomized primal coordinate-descent method of Nesterov [2010], 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 [Nesterov, 2010].
- **DCA**: Applying randomized coordinate ascent to the dual, with uniform example selection and an exact line-search [Shalev-Schwartz and Zhang, 2013b].

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 Figure 2. In these experiments we observe the following trends:

- **PCD vs. PCD-L**: For the problems with $n > p$ (top and bottom rows of Figure 2), there is little difference between uniform and Lipschitz sampling of the coordinates. For the problems with $p > n$ (middle row of Figure 2), sampling according to the Lipschitz constant leads to a large performance improvement over uniform sampling.
- **PCD vs. 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) vs. (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 Comparison of Step-Size Strategies

In our prior work we analyzed the step-sizes $\alpha_k = 1/2nL$ and $\alpha_k = 1/2n\mu$ [Le Roux et al., 2012], while Section 3 considers the choice $\alpha_k = 1/16L$ and Section 4.5 discusses the choices $\alpha_k = 1/L$ and $\alpha_k = 2/(L + n\mu)$ as in FG methods. In Figure 3 we compare the performance of these various

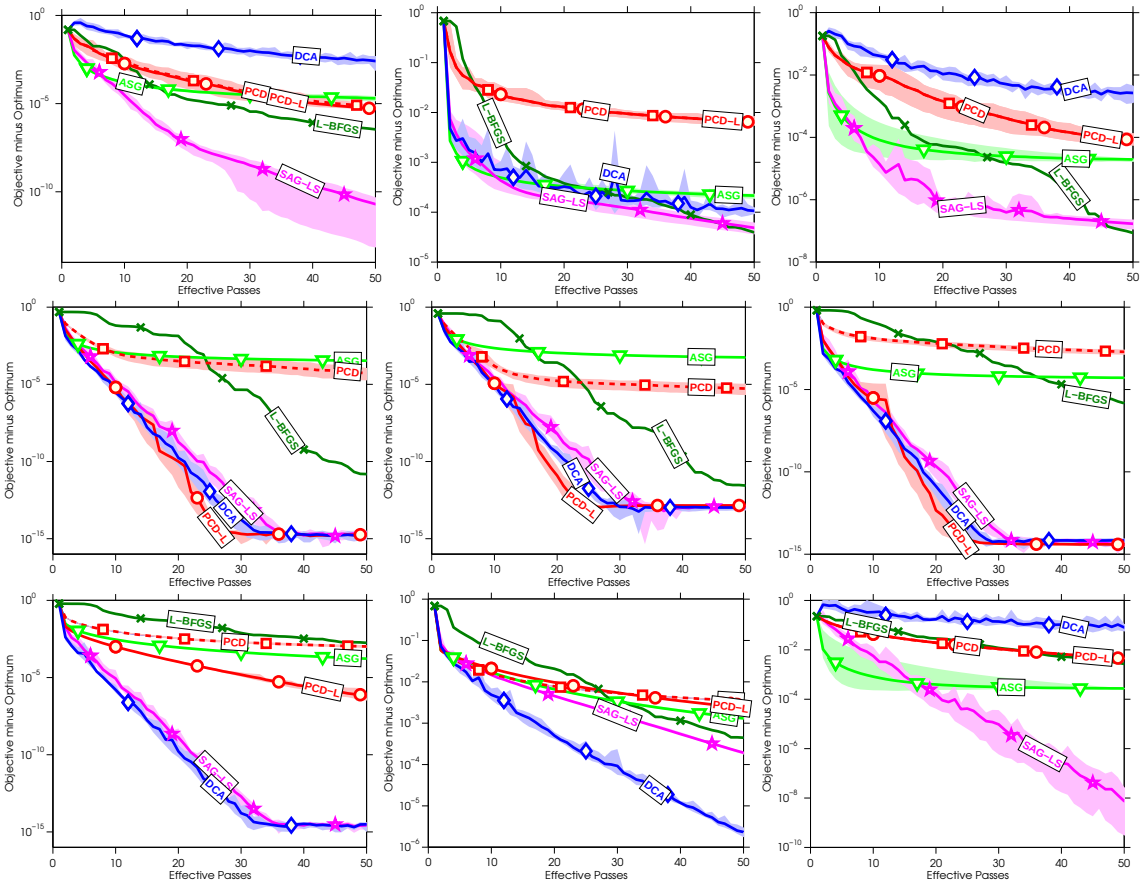

Figure 2: Comparison of optimization different FG and SG methods to coordinate optimization methods.The top row gives results on the *quantum* (left), *protein* (center) and *covertype* (right) datasets. The middle row gives results on the *rcv1* (left), *news* (center) and *spam* (right) datasets. The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

strategies to the performance of the SAG algorithm with our proposed line-search as well as the IAG and SAG algorithms when the best step-size is chosen in hindsight. In this plot we see the following trends:

- **Proposition 1 of Le Roux et al. [2012]**: Using a step-size of $\alpha_k = 1/2nL$ performs poorly, and makes little progress compared to the other methods. This makes sense because Proposition 1 in Le Roux et al. [2012] implies that the convergence rate (in terms of effective passes through the data) under this step size will be similar to the basic gradient method, which is known to perform poorly unless the problem is very well-conditioned.
- **Proposition 2 of Le Roux et al. [2012]**: Using a step-size of $\alpha_k = 1/2n\mu$ performs extremely well on the data sets with $p > n$ (middle row). In contrast, for the data sets with $n > p$ it often performs very poorly, and in some cases appears to diverge. This is consistent with Proposition 2 in Le Roux et al. [2012], which shows a fast convergence rate under this step size only if certain conditions on $\{n, \mu, L\}$ hold.
- **Theorem 1**: Using a step-size of $\alpha_k = 1/16L$ performs consistently better than the smaller step size $\alpha_k = 1/2nL$, but in some cases it performs worse than $\alpha_k = 1/2n\mu$. However, in contrast to $\alpha_k = 1/2n\mu$, the step size $\alpha_k = 1/16L$ always has reasonable performance.
- **Section 4.5**: The step size of $\alpha_k = 1/L$ performs performs extremely well on the data sets with $p > n$, and performs better than the step sizes discussed above on all but one of the remaining data sets. The step size of $\alpha_k = 2/(L + n\mu)$ seems to perform the same or slightly better than using $\alpha_k = 1/L$ except on one data set where it performs poorly.
- **Line-Search**: Using the line-search from Section 4.6 tends to perform as well or better than the various constant step size strategies, and tends to have similar performance to choosing the best step size in hindsight.
- **IAG vs. SAG**: When choosing the best step size in hindsight, the SAG iterations tend to choose a much larger step size than the IAG iterations. The step sizes chosen for SAG were 100 to 10000 times larger than the step sizes chosen by IAG, and always lead to better performance by several orders of magnitude.

5.4 Effect of mini-batches

As we discuss in Section 4.7, when using mini-batches within the SAG iterations there is a trade-off between the higher iteration cost of using mini-batches and the faster convergence rate obtained using mini-batches due to the possibility of using a smaller value of L. In Figure 4, we compare (on the dense data sets) the excess sub-optimality as a function of the number of examples seen for various mini-batch sizes and the three step-size strategies $1/L_{\max}$, $1/L_{\text{mean}}$, and $1/L_{\text{Hessian}}$ discussed in Section 4.7.

Several conclusions may be drawn from these experiments:

- Even though Theorem 1 hints at a maximum mini-batch size of $\frac{n\mu}{2L}$ without loss of convergence speed, this is a very conservative estimate. In our experiments, the original value of $\frac{n\mu}{L}$ was on the order of 10^{-5} and mini-batch sizes of up to 500 could be used without a loss in performance. Not only does this yield large memory storage gains, it would increase the computational efficiency of the algorithm when taking into account vectorization.

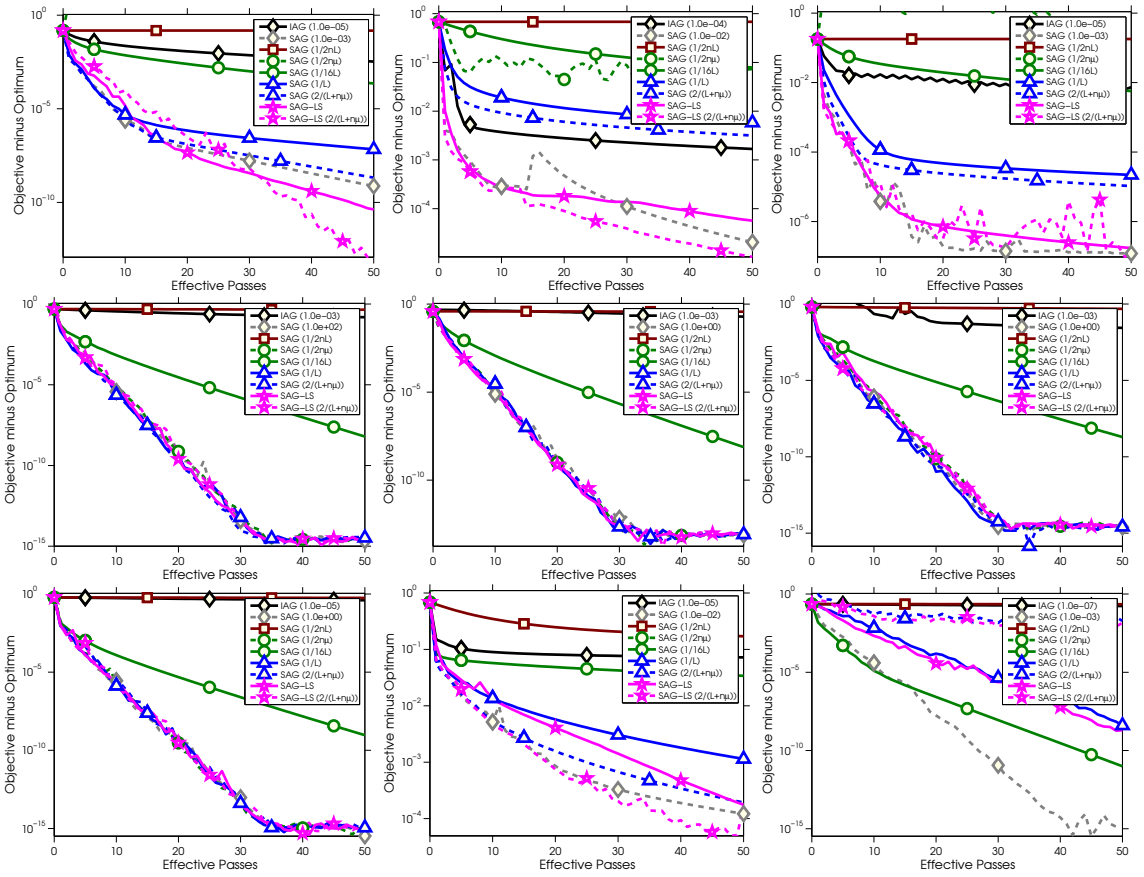

Figure 3: Comparison of step size strategies for the SAG method. The top row gives results on the *quantum* (left), *protein* (center) and *covertype* (right) datasets. The middle row gives results on the *rcv1* (left), *news* (center) and *spam* (right) datasets. The bottom row gives results on the *rcv1Full* (left), *sido* (center), and *alpha* (right) datasets. This figure is best viewed in colour.

- To achieve fast convergence, it is essential to use a larger step-size when larger mini-batches are used. For instance, in the case of the *quantum* dataset with a mini-batch size of 20000, we have $\frac{1}{L_{\max}} = 4.4 \cdot 10^{-4}$, $\frac{1}{L_{\text{mean}}} = 5.5 \cdot 10^{-2}$ and $\frac{1}{L_{\text{Hessian}}} = 3.7 \cdot 10^{-1}$. In the case of the *covertype* dataset and for a mini-batch size of 20000, we have $\frac{1}{L_{\max}} = 2.1 \cdot 10^{-5}$, $\frac{1}{L_{\text{mean}}} = 6.2 \cdot 10^{-2}$ and $\frac{1}{L_{\text{Hessian}}} = 4.1 \cdot 10^{-1}$.

5.5 Effect of non-uniform sampling

In our final experiment, we explored the effect of using the non-uniform sampling strategy discussed in Section 4.8. In Figure 5, 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 proporition 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 into the formula at the end of Section 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 $L_{\max}^k$ is computed in the same way as the SAG-LS method. To estimate each L_i, we keep track of an estimate L_i^k for each i and we set L_{mean}^k to the average of the L_i^k values among the f_i that we have sampled at least once. We set $L_i^k = L_i^{k-1}$ if example i was not selected and otherwise we initialize to $L_i^k = L_i^{k-1}/2$ and perform the line-search until we have a valid L_i^k (this means that L_{mean}^k 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 $L_i^k + L_{\text{mean}}^k$. 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 $\alpha_{\max} = 1/L_{\max}^k$ is the step-size we normally use with uniform sampling and $\alpha_{\text{mean}} = 1/2L_{\max}^k + 1/2L_{\text{mean}}^k$ 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.

We make the following observations from these experiments:

- **SAG (1/L) vs. 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 Figure 5).
- **SAG-LS vs. 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.

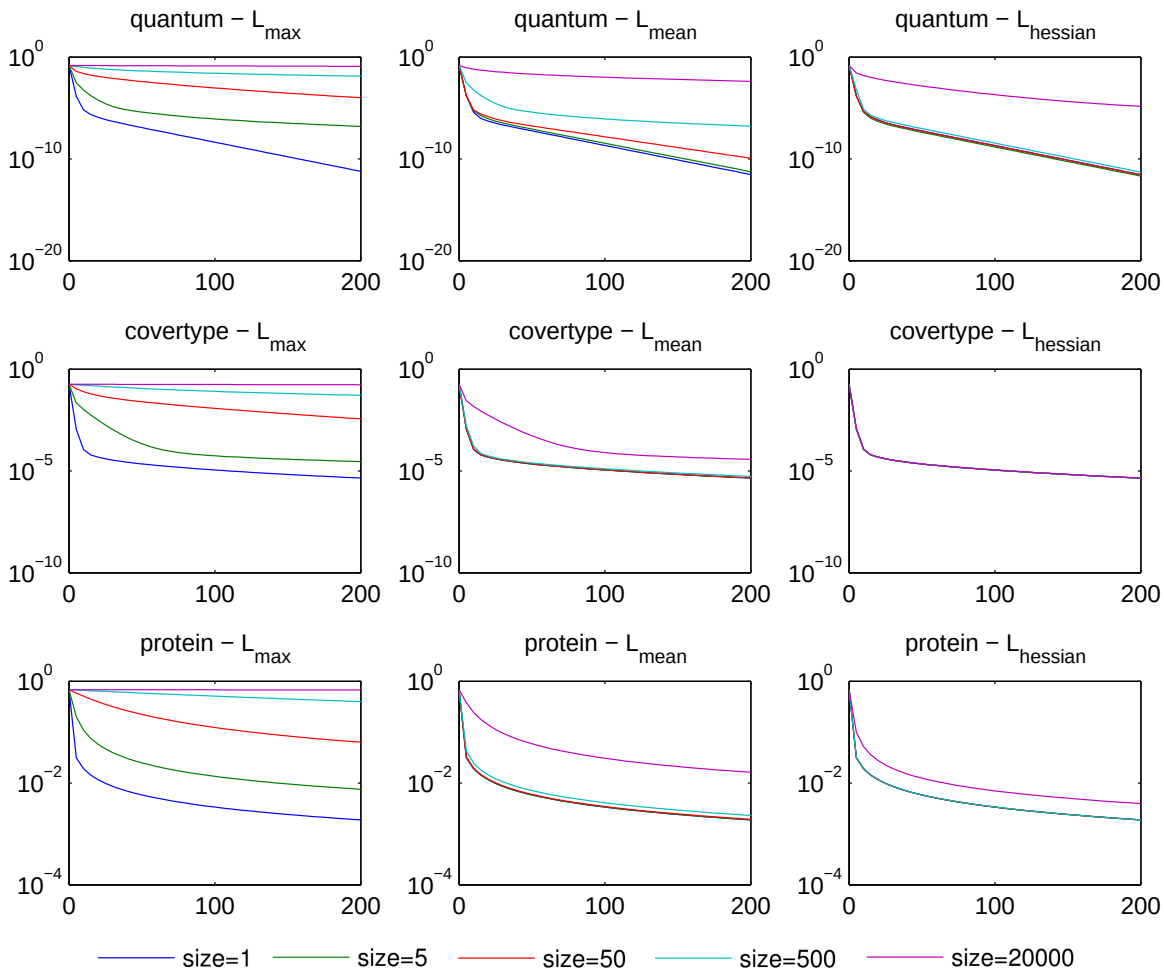

Figure 4: Sub-optimality as a function of the number of effective passes through the data for various datasets, step-size selection schemes and mini-batch sizes. The datasets are *quantum* (top), *covertype* (middle) and *protein* (bottom). Left: the step-size is $1/L$ with L the maximum Lipschitz constant of the individual gradients. It is thus the same for all mini-batch sizes. Center: the step-size is $1/L$ where L is obtained by taking the maximum among the averages of the Lipschitz constants within mini-batches. Right: the step-size is $1/L$ where L is obtained by computing the maximum eigenvalue of the Hessian for each mini-batch, then taking the maximum of these quantities across mini-batches.

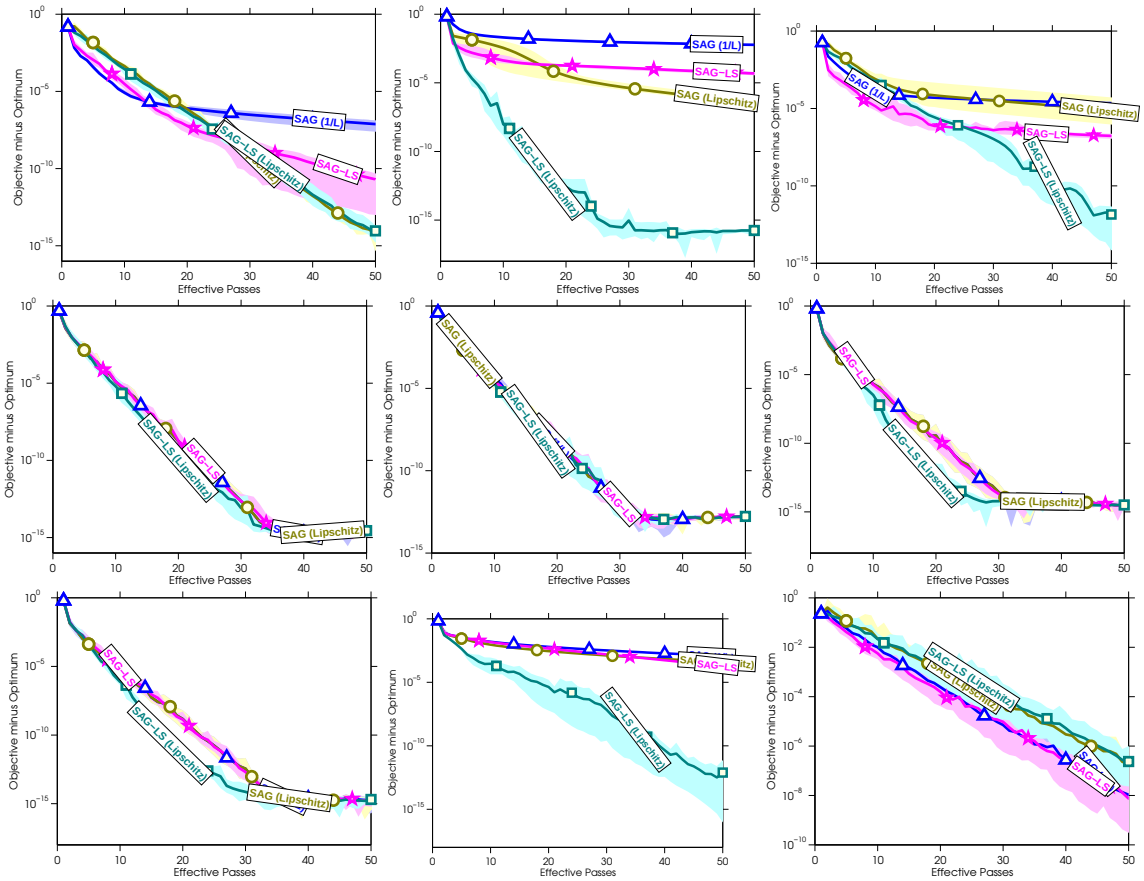

Figure 5: Comparison of uniform and non-uniform sampling strategies for the SAG algorithm. The top row gives results on the *quantum* (left), *protein* (center) and *covertype* (right) datasets. The middle row gives results on the *rcv1* (left), *news* (center) and *spam* (right) datasets. The bottom row gives results on the *rcv1Full*— (left), *sido* (center), and *alpha* (right) datasets. This figure is best viewed in colour.

6 Discussion

Since the first version of this work was published [Le Roux et al., 2012], there has been an explosion of interest in stochastic methods with improved convergence rates. In this section we first review other algorithms that have been discovered to have this property, and then we discuss the many possible variants on these basic algorithms that have been explored. As this is a very quickly-evolving area there are likely to be many new developments in the near future, but we note that this literature review is up to date as of January, 2015.

6.1 Alternative Algorithms

SDCA: The first algorithm that was shown to have a similar convergence rate and iteration cost to SAG was in fact a much older algorithm: coordinate optimization applied to a dual problem with randomized coordinate selection, referred to as stochastic dual coordinate ascent (SDCA). Shalev-Schwartz and Zhang [2013b] consider the problem of minimizing an ℓ_2-regularized finite sum,

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

where each f_i is convex and each f_i' is Lipschitz-continuous, by optimizing its Fenchel dual,

$$\underset{y\in\mathbb{R}^n}{\text{maximize}} -\frac{1}{2\lambda}\|\frac{1}{n}\sum_{i=1}^{n} y_i a_i\|^2 - \frac{1}{n}\sum_{i=1}^{n} f_i^*(-y_i).$$

They consider applying exact coordinate optimization to a randomly selected coordinate. Using the primal-dual relationship $x = \frac{1}{n\lambda}\sum_{i=1}^{N} x_i a_i$, they show a linear convergence rate in terms of the duality gap. Since there is one dual variable associated with each example i, the iteration cost is independent of n and thus the strategy has similar convergence properties to SAG (the memory requirements are identical in this context, see Section 4.1).

This result is related to the earlier result of [Nesterov, 2010], who shows a similar convergence rate for randomized coordinate descent.[7] However, the result of Nesterov cannot be directly applied to the dual problem in general since the dual does not necessarily have a Lipschitz-continuous gradient. A similar result was also reported even earlier by Collins et al. [2008] without requiring that the gradient of the dual is Lipschitz-continuous, but this result again only applied to the dual objective. More recently, Shalev-Schwartz and Zhang [2013a] show linear convergence of SDCA in the more general setting

$$\underset{x\in\mathbb{R}^p}{\text{minimize}}\, \lambda r(x) + \frac{1}{n}\sum_{i=1}^{n} f_i(A_i^T x),$$

where A_i are matrices and r is $1-$strongly convex. They also relax the requirement of exact coordinate optimization, providing a variety of more practical alternatives. Further, in subsequent work they obtain a convergence rate in the convex case by adding an explicit strongly-convex regularizer to the problem [Shalev-Schwartz and Zhang, 2014].

A disadvantage of SDCA compared to the SAG algorithm is that the SDCA convergence rates depend on λ rather than the strong-convexity constant μ. In the worst case we have $\mu = \lambda$, but if μ is much larger then the convergence rate of SAG is much faster. Further, even in cases where $\mu = \lambda$, the convergence rate of SAG might be much faster if the iterates stay in local region with a higher

[7] Local linear convergence rates of deterministic coordinate descent methods had been established much earlier [Luo and Tseng, 1992].

strong-convexity constant. As an extreme example, due to local strong-convexity SAG might have a linear convergence rate in scenarios where SDCA has a sub-linear convergence. This subtle but practically important issue was a key focus in the recent work of [Agarwal and Bottou, 2014], and indeed the performance of SDCA was very poor on three of the test problems in our experiments. See Figure 2 (top left, top right, bottom right).

MISO: Mairal [2013] analyzes a very general surrogate optimization framework, that includes a wide vareity of existing algorithms. He also considers incremental algorithms in this framework, and specialized to the smooth and unconstraeind setting (with a 'Lipschitz surrogate') obtains an algorithm (MISO) that is very similar to the SAG algorithm,

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

where y_i^k is defined as in the SAG algorithm 6, and x_i^k is the parameter vector used to compute the corresponding y_i^k. Thus, instead of applying the SAG step to x^k, MISO applies the step to the average of the previous x_i^k values used to form the current y_i^k variables. Mairal [2013] shows that this algorithm also achieves an $O(1/k)$ rate for convex objectives and a linear convergence rate for strongly-convex objectives.

However, MISO has the disadvantage that it not only requires storing the n gradient values but also storing n previous iterations (which are less likely to have a nice structure). Further, the linear convergence rate shown for MISO is substantially slower than the convergence rate shown in Theorem 1. In particular, the rate is more similar to the substantially slower Proposition 1 in our prior work [Le Roux et al., 2012]. Subsequent work on the MISO algorithm has shown an analogous result to Proposition 2 in our prior work; if the n is sufficiently larger than $L\mu$, then using a step-size proportional to $1/\mu$ yields a faster convergence rate [Mairal, 2014, Defazio et al., 2014b]. However, using this step-size causes divergence if μ is not sufficiently large.

SVRG: Another interesting framework that has been considered is known as 'mixed optimization', 'stochastic variance-reduced gradient' (SVRG), or 'semi-stochastic gradient descent' (S2GD) [Mahdavi and Jin, 2013, Johnson and Zhang, 2013, Zhang et al., 2013, Konečnỳ and Richtárik, 2013].[8] Unlike FG methods that utilize the full gradient g' and SG methods that consider individual gradients f_i', these mixed optimization methods combine the two. In particular, they use a (possibly regularized) iteration of the form

$$x^{k+1} = x^k - \alpha_k(f_i'(x^k) - f_i'(\tilde{x}^k) + \frac{1}{n}\sum_{j=1}^{n} f_j'(\tilde{x}^k)),$$

where $\tilde{x}^k$ is the last iterate where the full gradient g' was evaluated. The algorithm alternates between computing the full gradient at $\tilde{x}^k$, and performing some number m of stochastic gradient iterations. Note this is very similar to the SAG algorithm written in the form

$$x^{k+1} = x^k - \alpha_k(\frac{1}{n}f_i'(x^k) - \frac{1}{n}f_i'(x_i^k) + \frac{1}{n}\sum_{j=1}^{n} f_j'(x_i^k)),$$

where x_i^k is defined as in the MISO algorithm. These methods are thus similar to SAG in the use of potentially-outdated gradient information, but differ in that the outdated gradients are all computed at the same previous iteration $\tilde{x}^k$ and the weighting of terms is changed. If the step-size

[8] Wang et al. [2013] consider a related algorithm that maintains an easily-computable approximation to the current gradient $g'(x^k)$. Although this can improve the constants in the sublinear SG convergence rates, it does not improve the rates.

and parameter m are set appropriately, this algorithm has a linear convergence rate in the strongly-convex case. It can also achieve an $O(1/k)$ convergence rate in the general convex case, by applying it to a purturbed problem where a strongly-convex regularizer has been added.

A key advantage of this strategy is it only requires storing $\tilde{x}^k$, rather than the n gradient values. However, to obtain this improved memory requirement we must evaluate f_i' twice on each iteration. Further, the algorithm innefficiently requires full passes through the data to evaluate $f_j'(\tilde{x}^k)$ for all j.

A disadvantage of the SVRG method is that it requires setting two parameters (rather than one) and the convergence rate depends in a non-trivial way on their interaction both with each other and with both the Lipschitz constant and the strong-convexity. As with SDCA, the dependence on the strong-convexity constant is notable. In particular, the algorithm can't be applied without modification to general convex problems, and (although the effect is not as severe as it is with SDCA) the convergence rate may be substantially slower than SAG if there exists hidden strong convexity.

Self-Concordant Objectives: A surprising recent development is that Bach and Moulines [2013] have shown that the finite sum assumption is not required to obtain the $O(1/k)$ convergence rate in the special case of least squares. They show that an averaged SG method achieves this rate, and give a 2-phase Newton-like SG method that achieves this rate under an assumption similar to self-concordance. However, this assumption does not hold in general for the class of problems considered here.

SAGA: One of the most recent developments in this area is the SAGA algorithm Defazio et al. [2014a]. This algorithm intelligently combines the updates used in the SAG and SVRG algorithms. This method maintains the appealing properties of SAG, but yields a simpler proof. However, the proof technique used in that work does not yield a simpler analysis of the original SAG algorithm.

Lower Bounds: There has been recent work on determining a lower bound on the convergence rate that can be expected for minimizing finite sums. Defazio et al. [2014b] show that the rate must be at least $(1 - 1/n)$ in the strongly-convex case, while Agarwal and Bottou [2014] establish a bound that also depends on the condition number L/μ.

6.2 Generalizations and Other Issues

Accelerated gradient: AFG methods are variants of the basic FG method that can obtain an $O(1/k^2)$ convergence rate for the general convex case, and faster linear convergence rates that depend on the square root of the condition number ($\sqrt{L/\mu}$) rather than the condition number (L/μ) in the strongly-convex case [see Nesterov, 2004, §2.2.1]. For strongly-convex objectives, it has been shown that a mini-batch strategy can obtain a better dependence on the condition number in certain regimes for SDCA [Shalev-Shwartz and Zhang, 2013] and more recently for SVRG [Nitanda, 2014]. It is possible that similar arguments could hold for SAG algorithms, which could be advantageous over these methods for reasons discussed in the previous section.

Shalev-Schwartz and Zhang [2014] have also given an accelerated version of the SDCA method for ill-conditioned problems, that uses SDCA to solve a sequence of regularized problems up to a prescribed optimality. Since this procedure ultimately relies on the sequence of primal solutions, we can also accelerate SAG using a procedure like this. However, a difficulty with this procedure (whether SAG or SDCA are used) is the cost of measuring the optimality of the sub-problems.

Rather than using this inner-outer procedure, Lin et al. [2014] show that using a deterministic primal iteration (based on the full data set) allows one to construct a primal solution that has the accelerated rate from the result of an accelerated dual coordinate ascent method. Zhang and Xiao [2014] give a coordinate-wise variant of the accelerated primal-dual method of Chambolle and Pock

[2011] that also achieves this rate. Based on these results, it is possible that accelerated versions of SAG could be developed that do not rely on an inner-outer procedure.

The convergence rates of these accelerated methods have the same form as the lower bound established by Agarwal and Bottou [2014]. However, Agarwal and Bottou point out that these accelerated convergence rates depend on λ rather than μ. Thus, they conclude that the basic SAG algorithm may still be much faster on some problems than accelerated SDCA methods, and indeed show how to construct a problem where SAG is arbitrarily faster than accelerated SDCA. The possibility of developing an accelerated SAG algorithm that is adaptive to μ remains open.

Proximal gradient and ADMM: It is becoming increasingly common to address problems of the form

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

where f_i and g satisfy our assumptions and r is a general convex function that could be non-smooth or enforce that constraints are satisfied. Proximal-gradient methods for problems with this structure use iterations of the form

$$x^{k+1} = \text{prox}_{\alpha_k}\left[x^k - \frac{\alpha_k}{n}\sum_{i=1}^{n} f_i'(x^k)\right],$$

where the $\text{prox}_{\alpha_k}[y]$ operator is the solution to the proximity problem

$$\underset{x\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}\|x-y\|^2 + \alpha_k r(x).$$

Proximal-gradient and accelerated proximal-gradient methods are appealing for solving non-smooth optimization problems because they achieve the same convergence rates as FG methods for smooth optimization problems [Nesterov, 2007, Schmidt et al., 2011]. We have explored a variant of the proximal-gradient method where the average over the $f_i'(x^k)$ values is replaced by the SAG approximation (6). Although our analysis does not directly apply to this scenario, we believe that this proximal-SAG algorithm for composite non-smooth optimization achieves the same convergence rates as the SAG algorithm for smooth optimization; this is supported by the experiments of Xiao and Zhang [2014]. Indeed, there now exist proximal-gradient variants of SDCA [Shalev-Schwartz and Zhang, 2013a], MISO [Mairal, 2013], SVRG [Xiao and Zhang, 2014], and SAGA [Defazio et al., 2014a].

In cases where r is composed with a linear function, we can consider approaches based on the alternating direction method of multipliers (ADMM). This has been explored for SDCA [Suzuki, 2014] and MISO [Zhong and Kwok, 2014], and a variant based on SAG is also likely to be possible.

Coordinate-wise: The key advantage of SG and SAG methods is that the iteration cost is independent of the number of functions n. However, in many applications we may also be concerned with the dependence of the iteration cost on the number of variables p. Randomized coordinate-wise methods offer linear convergence rates with an iteration cost that is linear in n but independent of p [Nesterov, 2010]. We can consider a variant of SAG whose iteration cost is independent of both n and p by using the update

$$[y_i^k]_j = \begin{cases} [f_i'(x^k)]_j & \text{if } i = i_k \text{ and } j = j_k \\ [y_i^{k-1}]_j & \text{otherwise,} \end{cases}$$

to each coordinate j of each vector y_i in the SAG algorithm, and where j_k is a sample from the set $\{1, 2, \ldots, p\}$. Konečnỳ et al. [2014] recently proposed an SVRG algorithm of this flavour.

Newton-like: In cases where g is twice-differentiable, we can also consider Newton-like variants of the SAG algorithm,

$$x^{k+1} = x^k - \frac{\alpha_k}{n} B^k [\sum_{i=1}^{n} y_i^k],$$

where B^k is a positive-definite approximation to the inverse Hessian $g''(x^k)$. We would expect to obtain a faster convergence rate by using an appropriate choice of the sequence $\{B^k\}$. However, in order to not increase the iteration cost these matrices should be designed to allow fast multiplication. For example, we could choose B^k to be diagonal, which would also preserve any sparsity present in the updates. Sohl-Dickstein et al. [2014] propose a quasi-Newton method in this class that shows impressive empirical results, although the iteration cost is much higher.

Relaxing Convexity Assumptions: It is likely that the convexity assumptions made in this work could be relaxed. For example, Gong and Ye [2014] show SVRG can obtain a linear convergence under weaker assumptions. Since SAG is adaptive to hidden strong-convexity, the assumptions needed for its linear convergence are likely even weaker. Further, the SAG algorithm may also be useful even if g is non-convex (which is different than SDCA). In this case we expect that, similar to the IAG method [Blatt et al., 2007], the algorithm converges to a stationary point under very general conditions.

Non-Uniform Sampling: We have given an argument that non-uniform sampling should benefit the SAG algorithm, and shown empirically that it can lead to a substantial improvement. However, we have not yet given a full analysis of this scheme. Subsequent works have shown that the type of dependency we conjecture here (e.g., dependence on the average Lispschitz constant) can be achieved with non-uniform sampling in the context of SDCA [Qu et al., 2014, Zhao and Zhang, 2014], SVRG [Xiao and Zhang, 2014], and SAGA [Schmidt et al., 2015]

Step-size selection and termination criteria: 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 [see Nemirovski et al., 2009, §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.

Appendix A: Comparison of convergence rates

We now present a comparison of the convergence rates of primal and dual FG and coordinate-wise methods to the rate of the SAG method in terms of effective passes through the data for ℓ_2-regularized least-squares regression. In particular, in this appendix we will consider the problem

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

where to apply the SG or SAG method we can use

$$f_i(x) := \frac{\lambda}{2}\|x\|^2 + \frac{1}{2}(a_i^T x - b_i)^2.$$

If we use b to denote a vector containing the values b_i and A to denote a matrix withs rows a_i, we can re-write this problem as

$$\underset{x\in\mathbb{R}^p}{\text{minimize}} \quad \frac{\lambda}{2}\|x\|^2 + \frac{1}{2n}\|Ax - b\|^2.$$

The Fenchel dual of this problem is

$$\underset{y\in\mathbb{R}^n}{\text{minimize}} \quad d(y) := \frac{n}{2}\|y\|^2 + \frac{1}{2\lambda}y^\top AA^\top y + y^\top b.$$

We can obtain the primal variables from the dual variables by the formula $x = (-1/\lambda)A^\top y$. Convergence rates of different primal and dual algorithms are often expressed in terms of the following Lipschitz constants:

$$\begin{aligned}
L_g &= \lambda + M_\sigma/n && \text{(Lipschitz constant of } g'\text{)}\\
L_g^i &= \lambda + M_i && \text{(Lipschitz constant for all } f_i'\text{)}\\
L_g^j &= \lambda + M_j/n && \text{(Lipschitz constant of all } g_j'\text{)}\\
L_d &= n + M_\sigma/\lambda && \text{(Lipschitz constant of } d'\text{)}\\
L_d^i &= n + M_i/\lambda && \text{(Lipschitz constant of all } d_i'\text{)}
\end{aligned}$$

Here, we use M_σ to denote the maximum eigenvalue of $A^\top A$, M_i to denote the maximum squared row-norm $\max_i\{\|a_i\|^2\}$, and M_j to denote the maximum squared column-norm $\max_j\{\sum_{i=1}^n (a_i)_j^2\}$. We use g_j' to refer to element of j of g', and similarly for d_i'. The convergence rates will also depend on the primal and dual strong-convexity constants:

$$\begin{aligned}
\mu_g &= \lambda + m_\sigma/n && \text{(Strong-convexity constant of } g\text{)}\\
\mu_d &= n + m_\sigma'/\lambda && \text{(Strong-convexity constant of } d\text{)}
\end{aligned}$$

Here, m_σ is the minimum eigenvalue of $A^\top A$, and m_σ' is the minimum eigenvalue of $AA^\top$.

A.1 Full Gradient Methods

Using a similar argument to [Nesterov, 2004, Theorem 2.1.15], if we use the basic FG method with a step size of $1/L_g$, then $(f(x^k) - f(x^*))$ converges to zero with rate

$$\left(1 - \frac{\mu_g}{L_g}\right)^2 = \left(1 - \frac{\lambda + m_\sigma/n}{\lambda + M_\sigma/n}\right)^2 = \left(1 - \frac{n\lambda + m_\sigma}{n\lambda + M_\sigma}\right)^2 \leqslant \exp\left(-2\frac{n\lambda + m_\sigma}{n\lambda + M_\sigma}\right),$$

while a larger step-size of $2/(L_g + \mu_g)$ gives a faster rate of

$$\left(1 - \frac{\mu_g + \mu_g}{L_g + \mu_g}\right)^2 = \left(1 - \frac{n\lambda + m_\sigma}{n\lambda + (M_\sigma + m_\sigma)/2}\right)^2 \leqslant \exp\left(-2\frac{n\lambda + m_\sigma}{n\lambda + (M_\sigma + m_\sigma)/2}\right),$$

and we see that the speed improvement is determined by how much smaller m_σ is than M_σ.

If we use the basic FG method on the dual problem with a step size of $1/L_d$, then $(d(x^k) - d(x^*))$ converges to zero with rate

$$\left(1 - \frac{\mu_d}{L_d}\right)^2 = \left(1 - \frac{n + m'_\sigma/\lambda}{n + M_\sigma/\lambda}\right)^2 = \left(1 - \frac{n\lambda + m'_\sigma}{n\lambda + M_\sigma}\right)^2 \leqslant \exp\left(-2\frac{n\lambda + m'_\sigma}{n\lambda + M_\sigma}\right),$$

and with a step-size of $2/(L_d + \mu_d)$ the rate is

$$\left(1 - \frac{\mu_d + \mu_d}{L_d + \mu_d}\right)^2 = \left(1 - \frac{n\lambda + m'_\sigma}{n\lambda + (M_\sigma + m'_\sigma)/2}\right)^2 \leqslant \exp\left(-2\frac{n\lambda + m'_\sigma}{n\lambda + (M_\sigma + m'_\sigma)/2}\right).$$

Thus, whether we can solve the primal or dual method faster depends on m_σ and m'_σ. In the over-determined case where A has independent columns, a primal method should be preferred. In the under-determined case where A has independent rows, we can solve the dual more efficiently. However, we note that a convergence rate on the dual objective does not necessarily yield the same rate in the primal objective. If A is invertible (so that $m_\sigma = m'_\sigma$) or it has neither independent columns nor independent rows (so that $m_\sigma = m'_\sigma = 0$), then there is no difference between the primal and dual rates.

The AFG method achieves a faster rate. Applied to the primal with a step-size of $1/L_g$ it has a rate of [Nesterov, 2004, Theorem 2.2.2]

$$\left(1 - \sqrt{\frac{\mu_g}{L_g}}\right) = \left(1 - \sqrt{\frac{\lambda + m_\sigma/n}{\lambda + M_\sigma/n}}\right) = \left(1 - \sqrt{\frac{n\lambda + m_\sigma}{n\lambda + M_\sigma}}\right) \leqslant \exp\left(-\sqrt{\frac{n\lambda + m_\sigma}{n\lambda + M_\sigma}}\right),$$

and applied to the dual with a step-size of $1/L_d$ it has a rate of

$$\left(1 - \sqrt{\frac{\mu_d}{L_d}}\right) = \left(1 - \sqrt{\frac{n + m'_\sigma\lambda}{n + M_\sigma/\lambda}}\right) = \left(1 - \sqrt{\frac{n\lambda + m'_\sigma}{n\lambda + M_\sigma}}\right) \leqslant \exp\left(-\sqrt{\frac{n\lambda + m'_\sigma}{n\lambda + M_\sigma}}\right).$$

A.2 Coordinate-Descent Methods

The cost of applying one iteration of an FG method is $O(np)$. For this same cost we could apply p iterations of a coordinate descent method to the primal, assuming that selecting the coordinate to update has a cost of $O(1)$. If we select coordinates uniformly at random, then the convergence rate for p iterations of coordinate descent with a step-size of $1/L_g^j$ is [Nesterov, 2010, Theorem 2]

$$\left(1 - \frac{\mu_g}{pL_g^j}\right)^p = \left(1 - \frac{\lambda + m_\sigma/n}{p(\lambda + M_j/n)}\right)^p = \left(1 - \frac{n\lambda + m_\sigma}{p(n\lambda + M_j)}\right)^p \leqslant \exp\left(-\frac{n\lambda + m_\sigma}{n\lambda + M_j}\right).$$

Here, we see that applying a coordinate-descent method can be much more efficient than an FG method if $M_j << M_\sigma$. This can happen, for example, when the number of variables p is much larger than the number of examples n. Further, it is possible for coordinate descent to be faster than the AFG method if the difference between M_σ and M_j is sufficiently large.

For the $O(np)$ cost of one iteration of the FG method, we could alternately perform n iterations of coordinate descent on the dual problem. With a step size of $1/L_d^i$ this would obtain a rate on the dual objective of

$$\left(1-\frac{\mu_d}{nL_d^i}\right)^n = \left(1-\frac{n+m'_\sigma/\lambda}{n(n+M_i/\lambda)}\right)^n = \left(1-\frac{n\lambda+m'_\sigma}{n(n\lambda+M_i)}\right)^n \leqslant \exp\left(-\frac{n\lambda+m'_\sigma}{n\lambda+M_i}\right),$$

which will be faster than the dual FG method if $M_i << M_\sigma$. This can happen, for example, when the number of examples n is much larger than the number of variables p. The difference between the primal and dual coordinate methods depends on M_i compared to M_j and m_σ compared to m'_σ.

The analysis of SDCA gives a convergence rate in terms of the duality gap (and hence the primal) rather than simply in terms of the dual Shalev-Schwartz and Zhang [2013b]. Using the SDCA analysis, we obtain a rate in the duality gap of

$$\left(1-\frac{1}{n+M_i/\lambda}\right)^n = \left(1-\frac{\lambda}{n\lambda+M_i}\right)^n \leqslant \exp\left(-\frac{n\lambda}{n\lambda+M_i}\right),$$

which is the same as the dual rate given above when $m'_\sigma = 0$, and is slower otherwise.

A.3 Stochastic Average Gradient

For the $O(np)$ cost of one iteration of the FG method, we can perform n iterations of SAG. With a step size of $1/16L$, performing n iterations of the SAG algorithm has a rate of

$$\left(1-\min\left\{\frac{\mu_g}{16L_g^i},\frac{1}{8n}\right\}\right)^n = \left(1-\min\left\{\frac{\lambda+m_\sigma/n}{16(\lambda+M_i)},\frac{1}{8n}\right\}\right)^n \leqslant \exp\left(-\frac{1}{16}\min\left\{\frac{n\lambda+m_\sigma}{\lambda+M_i},2\right\}\right)$$

In the case where $n \leqslant 2L_g^i/\mu_g$, this is most similar to the rate obtained with the dual coordinate descent method, though there is a constant factor of 16 and the rate depends on the primal strong convexity constant m_σ rather than the dual m'_σ. However, in this case the SAG rate will often be faster because the term $n\lambda$ in the denominator is replaced by λ. An interesting aspect of the SAG rate is that unlike other methods the convergence rate of SAG reaches a limit: the convergence rate improves as n grows and as the condition number L_g^i/μ_g decreases but no further improvement in the convergence rate is obtained beyond the point where $n = 2L_g^i/\mu_g$. Thus, while SAG may not be the method of choice for very well-conditioned problems, it is a robust choice as it will always tend to be among the best methods in most situations.

Appendix B: Proof of the theorem

In this Appendix, we give the proof of Theorem 1.

B.1 Problem set-up and notation

Recall that we use $g = \frac{1}{n}\sum_{i=1}^n f_i$ to denote a μ-strongly convex function, where the functions f_i for $i = 1,\dots,n$ are convex functions from $\mathbb{R}^p$ to $\mathbb{R}$ with L-Lipschitz continuous gradients. In this appendix we will use the convention that $\mu \geqslant 0$ so that the regular convex case (where $\mu = 0$) is allowed. We assume that a minimizer of g is attained by some parameter x^* (such a value always exists and is unique when $\mu > 0$).

Recall that the SAG algorithm performs the recursion (for $k \geqslant 1$):

$$x^k \quad = \quad x^{k-1} - \frac{\alpha}{n} \sum_{i=1}^{n} y_i^k,$$

where an integer i_k is selected uniformly at random from $\{1, \ldots, n\}$ and we set

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

We will use the notation

$$y^k = \begin{pmatrix} y_1^k \\ \vdots \\ y_n^k \end{pmatrix} \in \mathbb{R}^{np}, \qquad \theta^k = \begin{pmatrix} y_1^k \\ \vdots \\ y_n^k \\ x^k \end{pmatrix} \in \mathbb{R}^{(n+1)p}, \qquad \theta^* = \begin{pmatrix} f_1'(x^*) \\ \vdots \\ f_n'(x^*) \\ x^* \end{pmatrix} \in \mathbb{R}^{(n+1)p},$$

and we will also find it convenient to use

$$e = \begin{pmatrix} I \\ \vdots \\ I \end{pmatrix} \in \mathbb{R}^{np \times p}, \qquad f'(x) = \begin{pmatrix} f_1'(x) \\ \vdots \\ f_n'(x) \end{pmatrix} \in \mathbb{R}^{np}. \tag{11}$$

With this notation, note that $g'(x) = \frac{1}{n} e^\top f'(x)$ and $x^k = x^{k-1} - \frac{\alpha}{n} e^\top y^k$.

For a square $n \times n$ matrix M, we use $\mathrm{diag}(M)$ to denote a vector of size n composed of the diagonal of M, while for a vector m of dimension n, $\mathrm{Diag}(m)$ is the $n \times n$ diagonal matrix with m on its diagonal. Thus $\mathrm{Diag}(\mathrm{diag}(M))$ is a diagonal matrix with the diagonal elements of M on its diagonal, and $\mathrm{diag}(\mathrm{Diag}(m)) = m$.

In addition, if M is a $tp \times tp$ matrix and m is a $tp \times p$ matrix, then we will use the convention that:

- $\mathrm{diag}(M)$ is the $tp \times p$ matrix being the concatenation of the t $(p \times p)$-blocks on the diagonal of M;
- $\mathrm{Diag}(m)$ is the $tp \times tp$ block-diagonal matrix whose $(p \times p)$-blocks on the diagonal are equal to the $(p \times p)$-blocks of m.

Finally, $\mathcal{F}_k$ will denote the σ-field of information up to (and including) time k. In other words, $\mathcal{F}_k$ is the σ-field generated by $i_1, \ldots, i_k$. Given $\mathcal{F}_{k-1}$, we can write the expected values of the y^k and x^k variables in the SAG algorithm as

$$\begin{aligned} \mathbb{E}(y^k | \mathcal{F}_{k-1}) \quad &= \quad \big(1 - \frac{1}{n}\big) y^{k-1} + \frac{1}{n} f'(x^{k-1}), \\ \mathbb{E}(x^k | \mathcal{F}_{k-1}) \quad &= \quad x^{k-1} - \frac{\alpha}{n}\big(1 - \frac{1}{n}\big) e^\top y^{k-1} - \frac{\alpha}{n^2} e^\top f'(x^{k-1}). \end{aligned}$$

The proof is based on finding a Lyapunov function $\mathcal{L}$ from $\mathbb{R}^{(n+1)p}$ to $\mathbb{R}$ such that the sequence $\mathbb{E}\mathcal{L}(\theta^k)$ decreases at an appropriate rate, and $\mathbb{E}\mathcal{L}(\theta^k)$ dominates $[g(x^k) - g(x^*)]$. We derive these results in a parameterized way, leaving a variety of coefficients undetermined but tracking the constraints required of the coefficients as we go. Subsequently, we guide the setting of these coefficients by using a second-order cone solver, and verify the validity of the resulting coefficients using a symbolic solver

to check positivity of certain polynomials. Finally, the constants in the convergence rate are given by the initial values of the Lyapunov function, based on the choice of y^0.

The Lyapunov function contains a term of the form $(\theta^k - \theta^*)^\top \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} (\theta^k - \theta^*)$ for some values of A, B and C. Our analysis makes use of the following lemma, derived in [Le Roux et al., 2012, Appendix A.4], showing how this quadratic form evolves through the SAG recursion in terms of the elements of θ^{k-1}.

Lemma 1. *If* $P = \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix}$, *for* $A \in \mathbb{R}^{np \times np}$, $B \in \mathbb{R}^{np \times p}$ *and* $C \in \mathbb{R}^{p \times p}$, *then*

$$\begin{aligned}
&\mathbb{E}\left[(\theta^k - \theta^*)^\top \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} (\theta^k - \theta^*) \middle| \mathcal{F}_{k-1}\right] \\
&\qquad = (y^{k-1} - f'(x^*))^\top \left[\left(1 - \frac{2}{n}\right) S + \frac{1}{n} \operatorname{Diag}(\operatorname{diag}(S))\right] (y^{k-1} - f'(x^*)) \\
&\qquad + \frac{1}{n} (f'(x^{k-1}) - f'(x^*))^\top \operatorname{Diag}(\operatorname{diag}(S)) (f'(x^{k-1}) - f'(x^*)) \\
&\qquad + \frac{2}{n} (y^{k-1} - f'(x^*))^\top [S - \operatorname{Diag}(\operatorname{diag}(S))] (f'(x^{k-1}) - f'(x^*)) \\
&\qquad + 2\left(1 - \frac{1}{n}\right) (y^{k-1} - f'(x^*))^\top \left[B - \frac{\alpha}{n} eC\right] (x^{k-1} - x^*) \\
&\qquad + \frac{2}{n} (f'(x^{k-1}) - f'(x^*))^\top \left[B - \frac{\alpha}{n} eC\right] (x^{k-1} - x^*) \\
&\qquad + (x^{k-1} - x^*)^\top C (x^{k-1} - x^*) \,,
\end{aligned}$$

with

$$S = A - \frac{\alpha}{n} B e^\top - \frac{\alpha}{n} e B^\top + \frac{\alpha^2}{n^2} e C e^\top \in \mathbb{R}^{np \times np} \,.$$

Our proof also uses the following lemma, giving the inverse of a highly-structured matrix that arises in the analysis.

Lemma 2. *Let* I *be the identity in* $\mathbb{R}^{np \times np}$ *and* $e \in \mathbb{R}^{np \times p}$ *be defined by stacking identity matrices in* $\mathbb{R}^{n \times n}$ *as in* (11). *If* α *and* β *are non-zero real-valued scalars, then it holds that*

$$\left(\alpha\left(I - \frac{1}{n} ee^T\right) + \beta\left(\frac{1}{n} ee^T\right)\right)^{-1} = \frac{1}{\alpha}\left(I - \frac{1}{n} ee^T\right) + \frac{1}{\beta}\left(\frac{1}{n} ee^T\right).$$

Proof. It is sufficient to verify that the inverse of the left side times the right side is equal to the identity matrix,

$$\begin{aligned}
&\left(\alpha\left(I - \frac{1}{n} ee^T\right) + \beta\left(\frac{1}{n} ee^T\right)\right)\left(\frac{1}{\alpha}\left(I - \frac{1}{n} ee^T\right) + \frac{1}{\beta}\left(\frac{1}{n} ee^T\right)\right) \\
&= \left(I - \frac{1}{n} ee^T\right)\left(I - \frac{1}{n} ee^T\right) + \frac{\alpha}{\beta}\left(I - \frac{1}{n} ee^T\right)\left(\frac{1}{n} ee^T\right) \\
&+ \frac{\beta}{\alpha}\left(\frac{1}{n} ee^T\right)\left(I - \frac{1}{n} ee^T\right) + \left(\frac{1}{n} ee^T\right)\left(\frac{1}{n} ee^T\right) \\
&= \left(I - \frac{2}{n} ee^T + \frac{1}{n} ee^T\right) + \frac{\alpha}{\beta}\left(\frac{1}{n} ee^T - \frac{1}{n} ee^T\right) + \frac{\beta}{\alpha}\left(\frac{1}{n} ee^T - \frac{1}{n} ee^T\right) + \frac{1}{n} ee^T \\
&= I.
\end{aligned}$$

where we use that $e^T e = nI$, giving $\left(\frac{1}{n} ee^T\right)\left(\frac{1}{n} ee^T\right) = \frac{1}{n} ee^T$. □

B.2 General Lyapunov function

For some $h \geqslant 0$, we consider a Lyapunov function of the form

$$\mathcal{L}(\theta^k) = 2hg(x^k + de^\top y^k) - 2hg(x^*) + (\theta^k - \theta^*)^\top \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} (\theta^k - \theta^*)$$

with

$$A = a_1 ee^\top + a_2 I, \quad B = be, \quad C = cI.$$

To achieve the desired convergence rate, our goal is to show for appropriate values of $\delta \geqslant 0$ and $\gamma \geqslant 0$ that

$$\begin{aligned} &(a) \qquad \mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) \leqslant (1-\delta)\mathcal{L}(\theta^{k-1}), \\ &(b) \qquad \mathcal{L}(\theta^k) \geqslant \gamma\big[g(x^k) - g(x^*)\big], \end{aligned}$$

almost surely. Thus, in addition to the algorithm parameter α, there are 2 parameters of the result $\{\gamma, \delta\}$ and 6 parameters of the Lyapunov function $\{a_1, a_2, b, c, d, h\}$.

B.3 Lyapunov upper bound

To show (a), we derive an upper bound on the quantity

$$\begin{aligned} &\mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) - (1-\delta)\mathcal{L}(\theta^{k-1}) \\ &= 2h\mathbb{E}\big[g(x^k + de^\top y^k)\big] - 2hg(x^*) - (1-\delta)(2hg(x^{k-1} + de^\top y^{k-1}) - 2hg(x^*)) \\ &+\mathbb{E}\big[(\theta^k - \theta^*)^\top \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} (\theta^k - \theta^*)\big] - (1-\delta)(\theta^{k-1} - \theta^*)^\top \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} (\theta^{k-1} - \theta^*). \end{aligned}$$

Lemma 1 gives an expression for the expectation over the quadratic term in the last line, and we will have

$$\begin{aligned} S &= a_2 I + \left(a_1 - 2\frac{\alpha}{n}b + \frac{\alpha^2}{n^2}c\right)ee^\top \\ \mathrm{Diag}(\mathrm{diag}(S)) &= \left(a_2 + a_1 - 2\frac{\alpha}{n}b + \frac{\alpha^2}{n^2}c\right)I \\ S - \mathrm{Diag}(\mathrm{diag}(S)) &= \left(a_1 - 2\frac{\alpha}{n}b + \frac{\alpha^2}{n^2}c\right)(ee^\top - I). \end{aligned}$$

We also use that strong convexity of g implies

$$2g(x^{k-1} + de^\top y^{k-1}) \quad \geqslant \quad 2g(x^{k-1}) + 2dg'(x^{k-1})^\top e^\top y^{k-1} + \mu d^2 \|e^\top y^{k-1}\|^2. \tag{12}$$

Further, using the Lipschitz-continuity of g' and the identity $x^k + de^\top y^k = x^{k-1} + (d - \frac{\alpha}{n})e^\top y^k$, followed by applying Lemma 1 using $A = ee^\top$ to expand the last term, gives us the bound

$$\begin{aligned}
&\mathbb{E}\big[2g(x^k + de^\top y^k)|\mathcal{F}_{k-1}\big] \\
&\leqslant 2g(x^{k-1}) + 2\big(d - \frac{\alpha}{n}\big)g'(x^{k-1})^\top \mathbb{E}\big[e^\top y^k|\mathcal{F}_{k-1}\big] + L\big(d - \frac{\alpha}{n}\big)^2\mathbb{E}\big[\|e^\top y^k\|^2|\mathcal{F}_{k-1}\big] \\
&= 2g(x^{k-1}) + 2\big(d - \frac{\alpha}{n}\big)g'(x^{k-1})^\top\big[(1 - \frac{1}{n})e^\top y^{k-1} + \frac{1}{n}e^\top f'(x^{k-1})\big] \\
&+ L\big(d - \frac{\alpha}{n}\big)^2\left[(y^{k-1} - f'(x^*))^\top\left[\left(1 - \frac{2}{n}\right)ee^\top + \frac{1}{n}I\right](y^{k-1} - f'(x^*))\right] \qquad (13) \\
&+ L\big(d - \frac{\alpha}{n}\big)^2\left[\frac{1}{n}\|f'(x^{k-1}) - f'(x^*)\|^2\right. \\
&\qquad\qquad \left. + \frac{2}{n}(y^{k-1} - f'(x^*))^\top\left[ee^\top - I\right](f'(x^{k-1}) - f'(x^*))\right].
\end{aligned}$$

Combining Lemma 1 with Inequalities (12) and (13) yields an upper bound on $\mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) - (1-\delta)\mathcal{L}(\theta^{k-1})$ with a large number of terms. To help in the process of simplifying these terms, Table B.3 lists all the terms up to a scalar factor s and possibly a matrix M. The table also gives these scalars and matrices for each term, as well as the source of the term.

We combine the expressions from Table B.3 using the stated groups in the last column. For example, we add together all the terms in group 0 to obtain a scalar coefficient B_0 for terms in this group. We do this in the straightforward way (using that $g'(x) = \frac{1}{n}e^\top f'(x)$ and $g'(x^*) = 0$) for groups 0, 1, 2, 5, 8, and 9. For groups 6 and 7 we split into terms that have an identity matrix M (B_6) and terms with an $ee^\top$ term (B_7). Similarly, B_3 comes from terms with an identity matrix in M and B_4 correspond to terms with an $ee^\top$ term. Combining expressions in this way (and adding/subtracting $(B_3/n)ee^\top$) gives:

$$\begin{aligned}
&\mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) - (1-\delta)\mathcal{L}(\theta^{k-1}) \\
\leqslant\ & B_0\big(g(x^{k-1}) - g(x^*)\big) + B_9\|x^{k-1} - x^*\|^2 + (x^{k-1} - x^*)^\top g'(x^{k-1})B_1 - B_2\|g'(x^{k-1})\|^2 \\
& -(y^{k-1} - y^*)^\top\left[B_3(I - \frac{1}{n}ee^\top) + B_4\frac{1}{n}ee^\top\right](y^{k-1} - y^*) \\
& +(y^{k-1} - y^*)^\top\left[B_5e(x^{k-1} - x^*) + B_6(f'(x^{k-1}) - f'(x^*)) + B_7eg'(x^{k-1})\right] \\
& +B_8\|f'(x^{k-1}) - f'(x^*)\|^2,
\end{aligned}$$

Term	Scalar s	Matrix M	Source	Group
$sg(x^*)$	$-2h$		$\mathbb{E}\big(\mathcal{L}(\theta^k)\vert\mathcal{F}_{k-1}\big)$	0
$sg(x^*)$	$2h(1-\delta)$		$L(\theta^{k-1})$	0
$s(y^{k-1}-f'(x^*))^\top M(y^{k-1}-f'(x^*))$	$-(1-\delta)$	$a_1 ee^\top + a_2 I$	$L(\theta^{k-1})$	3 and 4
$s(y^{k-1}-f'(x^*))^\top M(x^{k-1}-x^*)$	$-2(1-\delta)$	be	$L(\theta^{k-1})$	5
$s(x^{k-1}-x^*)\top M(x^{k-1}-x^*)$	$-(1-\delta)$	cI	$L(\theta^{k-1})$	9
$sg(x^{k-1})$	$-2h(1-\delta)$		Inequality (12)	0
$sg'(x^{k-1})^\top My^{k-1}$	$-2h(1-\delta)d$	$e^\top$	Inequality (12)	7
$s(y^{k-1})^\top My^{k-1}$	$-h(1-\delta)\mu d^2$	$ee^\top$	Inequality (12)	4
$sg(x^{k-1})$	$2h$		Inequality (13)	0
$sg'(x^{k-1})^\top My^{k-1}$	$2h(d-\frac{\alpha}{n})$	$(1-\frac{1}{n})e^\top$	Inequality (13)	7
$sg'(x^{k-1})^\top Mf'(x^{k-1})$	$2h(d-\frac{\alpha}{n})$	$\frac{1}{n}e^\top$	Inequality (13)	2
$s(y^{k-1}-f'(x^*))^\top M(y^{k-1}-f'(x^*))$	$Lh(d-\frac{\alpha}{n})^2$	$\left(1-\frac{2}{n}\right)ee^\top+\frac{1}{n}I$	Inequality (13)	3 and 4
$s(f'(x^{k-1})-f'(x^*))^\top M(f'(x^{k-1})-f'(x^*))$	$\frac{Lh}{n}(d-\frac{\alpha}{n})^2$	I	Inequality (13)	8
$s(y^{k-1}-f'(x^*))^\top M(f'(x^{k-1})-f'(x^*))$	$\frac{2Lh}{n}(d-\frac{\alpha}{n})^2$	$ee^\top - I$	Inequality (13)	6 and 7
$s(y^{k-1}-f'(x^*))^\top M(y^{k-1}-f'(x^*))$	1	$\left(1-\frac{2}{n}\right)S+\frac{1}{n}\operatorname{Diag}(\operatorname{diag}(S))$	Lemma 1	3 and 4
$s(y^{k-1}-f'(x^*))^\top M(x^{k-1}-x^*)$	$2\left(1-\frac{1}{n}\right)$	$(b-\frac{\alpha}{n}c)e$	Lemma 1	5
$s(x^{k-1}-x^*)\top M(x^{k-1}-x^*)$	1	cI	Lemma 1	9
$s(f'(x^{k-1})-f'(x^*))^\top M(f'(x^{k-1})-f'(x^*))$	$\frac{1}{n}$	$\operatorname{Diag}(\operatorname{diag}(S))$	Lemma 1	8
$s(y^{k-1}-f'(x^*))^\top M(f'(x^{k-1})-f'(x^*))$	$\frac{1}{n}$	$S-\operatorname{Diag}(\operatorname{diag}(S))$	Lemma 1	6 and 7
$s(f'(x^{k-1})-f'(x^*))^\top M(x^{k-1}-x^*)$	$\frac{2}{n}$	$(b-\frac{\alpha}{n}c)e$	Lemma 1	1

Table 3: Expressions in upper bound on $\mathbb{E}\big(\mathcal{L}(\theta^k)\vert\mathcal{F}_{k-1}\big)-(1-\delta)\mathcal{L}(\theta^{k-1})$.

with

$$
\begin{aligned}
B_0 &= 2\delta h \\
B_1 &= 2(b-\frac{\alpha}{n}c) \\
B_2 &= 2(\frac{\alpha}{n}-d)h \\
B_3 &= -\Big[\Big(1-\frac{2}{n}\Big)a_2+\frac{1}{n}\big[a_1+a_2-2\frac{\alpha}{n}b+\frac{\alpha^2}{n^2}c\big]-(1-\delta)a_2+Lh\frac{1}{n}\big(d-\frac{\alpha}{n}\big)^2\Big] \\
B_4 &= B_3 \\
&- n\Big[\Big(1-\frac{2}{n}\Big)(a_1-2\frac{\alpha}{n}b+\frac{\alpha^2}{n^2}c)-(1-\delta)a_1+L(1-\frac{2}{n})h\big(d-\frac{\alpha}{n}\big)^2-(1-\delta)\mu h d^2\Big] \\
B_5 &= 2\Big[(\delta-\frac{1}{n})b-\frac{\alpha}{n}(1-\frac{1}{n})c\Big] \\
B_6 &= -\frac{2}{n}\Big(hL\big(d-\frac{\alpha}{n}\big)^2+a_1-\frac{2\alpha}{n}b+\frac{\alpha^2}{n^2}c\Big) \\
B_7 &= \Big(\Big[2\Big(hL\big(d-\frac{\alpha}{n}\big)^2+a_1-\frac{2\alpha}{n}b+\frac{\alpha^2}{n^2}c\Big)+2\Big(h(d-\frac{\alpha}{n})(1-\frac{1}{n})-h(1-\delta)d\Big)\Big]\Big) \\
B_8 &= \Big[\frac{1}{n}\big(a_1+a_2-2\frac{\alpha}{n}b+\frac{\alpha^2}{n^2}c\big)+\frac{L}{n}h\big(d-\frac{\alpha}{n}\big)^2\Big] \\
B_9 &= c\delta.
\end{aligned}
$$

In this expression, $y^{k-1}-y^*$ only appears through a quadratic form, with a quadratic term $-(y^{k-1}-y^*)^\top\Big[B_3(I-\frac{1}{n}ee^\top)+B_4\frac{1}{n}ee^\top\Big](y^{k-1}-y^*)$. If $B_3 \geqslant 0$ and $B_4 \geqslant 0$, then this quadratic term is non-positive and we may maximize in closed form with respect to y^{k-1} to obtain an upper bound. Assuming that $B_3 \neq 0$ and $B_4 \neq 0$, we use Lemma 2, $\Big[B_3(I-\frac{1}{n}ee^\top)+B_4\frac{1}{n}ee^\top\Big]^{-1} = \Big[B_3^{-1}(I-\frac{1}{n}ee^\top)+B_4^{-1}\frac{1}{n}ee^\top\Big]$, to obtain:

$$
\begin{aligned}
&\mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big)-(1-\delta)\mathcal{L}(\theta^{k-1}) \\
\leqslant\ & B_0\big(g(x^{k-1})-g(x^*)\big)+B_9\|x^{k-1}-x^*\|^2+(x^{k-1}-x^*)^\top g'(x^{k-1})B_1-B_2\|g'(x^{k-1})\|^2 \\
&+\frac{1}{4}\frac{B_6^2}{B_3}(f'(x^{k-1})-f'(x^*))^\top(I-\frac{1}{n}ee^\top)(f'(x^{k-1})-f'(x^*)) \\
&+\frac{1}{4}\frac{1}{B_4}\Big[B_5e(x^{k-1}-x^*)+B_6(f'(x^{k-1})-f'(x^*))+B_7eg'(x^{k-1})\Big]^\top(\frac{1}{n}ee^\top) \\
&\qquad\qquad\Big[B_5e(x^{k-1}-x^*)+B_6(f'(x^{k-1})-f'(x^*))+B_7eg'(x^{k-1})\Big] \\
&+B_8\|f'(x^{k-1})-f'(x^*)\|^2 \\
=\ & B_0\big(g(x^{k-1})-g(x^*)\big)+B_9\|x^{k-1}-x^*\|^2+(x^{k-1}-x^*)^\top g'(x^{k-1})B_1-B_2\|g'(x^{k-1})\|^2 \\
&+\frac{1}{4}\frac{B_6^2}{B_3}(f'(x^{k-1})-f'(x^*))^\top(I-\frac{1}{n}ee^\top)(f'(x^{k-1})-f'(x^*)) \\
&+\frac{n}{4}\frac{1}{B_4}\big\|B_5(x^{k-1}-x^*)+\big(B_7+B_6\big)g'(x^{k-1})\big\|^2 \\
&+B_8\|f'(x^{k-1})-f'(x^*)\|^2.
\end{aligned}
$$

By using Lipschitz continuity of each f_i' to observe that $\|f'(x^{k-1}) - f'(x^*)\|^2 = \sum_{i=1}^n \|f_i'(x^{k-1}) - f_i'(x^*)\|^2 \leqslant L\sum_{i=1}^n (f_i'(x^{k-1}) - f_i'(x^*))^\top (x^{k-1} - x^*) = nLg'(x^{k-1})^\top (x^{k-1} - x^*)$, we obtain:

$$
\begin{aligned}
& \mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) - (1-\delta)\mathcal{L}(\theta^{k-1}) \\
\leqslant\ & B_0\left[(x^{k-1}-x^*)^\top g'(x_{k-1}) - \frac{\mu}{2}\|x^{k-1}-x^*\|^2\right] + B_9\|x^{k-1}-x^*\|^2 \\
& + (x^{k-1}-x^*)^\top g'(x^{k-1})B_1 - B_2\|g'(x^{k-1})\|^2 \\
& + \left(\frac{nL}{4}\frac{B_6^2}{B_3} + nLB_8\right)(x^{k-1}-x^*)^\top g'(x^{k-1}) - \frac{n}{4}\frac{B_6^2}{B_3}\|g'(x^{k-1})\|^2 \\
& + \frac{n}{4}\frac{B_5^2}{B_4}\|x^{k-1}-x^*\|^2 + \frac{n}{4}\frac{(B_6+B_7)^2}{B_4}\|g'(x^{k-1})\|^2 \\
& + \frac{2n}{4}\frac{B_5(B_6+B_7)}{B_4}(x^{k-1}-x^*)^\top g'(x^{k-1}) \\
=\ & \|x^{k-1}-x^*\|^2\left[-B_0\frac{\mu}{2} + B_9 + \frac{n}{4}\frac{B_5^2}{B_4}\right] \\
& + (x^{k-1}-x^*)^\top g'(x^{k-1})\left[B_0 + B_1 + \frac{nL}{4}\frac{B_6^2}{B_3} + \frac{2n}{4}\frac{B_5(B_6+B_7)}{B_4} + nLB_8\right] \\
& + \|g'(x^{k-1})\|^2\left[-B_2 - \frac{n}{4}\frac{B_6^2}{B_3} + \frac{n}{4}\frac{(B_6+B_7)^2}{B_4}\right] \\
=\ & C_0\|x^{k-1}-x^*\|^2 + C_1(x^{k-1}-x^*)^\top g'(x^{k-1}) + C_2\|g'(x^{k-1})\|^2,
\end{aligned}
$$

with

$$
\begin{aligned}
C_0 &= -B_0\frac{\mu}{2} + B_9 + \frac{n}{4}\frac{B_5^2}{B_4} \\
C_1 &= B_0 + B_1 + \frac{nL}{4}\frac{B_6^2}{B_3} + \frac{2n}{4}\frac{B_5(B_6+B_7)}{B_4} + nLB_8 \\
C_2 &= -B_2 - \frac{n}{4}\frac{B_6^2}{B_3} + \frac{n}{4}\frac{(B_6+B_7)^2}{B_4}.
\end{aligned}
$$

In order to show the decrease of the Lyapunov function, we need to show that $C_0\|x^{k-1}-x^*\|^2 + C_1(x^{k-1}-x^*)^\top g'(x^{k-1}) + C_2\|g'(x^{k-1})\|^2 \leqslant 0$ for all x^{k-1}.

If we assume that $C_1 \leqslant 0$ and $C_2 \leqslant 0$, then we have by strong-convexity

$$
C_0\|x^{k-1}-x^*\|^2 + C_1(x^{k-1}-x^*)^\top g'(x^{k-1}) + C_2\|g'(x^{k-1})\|^2 \leqslant \|x^{k-1}-x^*\|^2\big(C_0 + \mu C_1 + \mu^2 C_2\big),
$$

and thus the condition is that

$$
C_0 + \mu C_1 + \mu^2 C_2 \leqslant 0.
$$

And if we want to show that $\mathbb{E}\big(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}\big) - (1-\delta)\mathcal{L}(\theta^{k-1}) \leqslant -C_3(x^{k-1}-x^*)^\top g'(x^{k-1})$, then it suffices to have $C_1 + C_3 \leqslant 0$ and $C_0 + \mu(C_1+C_3) + \mu^2 C_2 \leqslant 0$.

B.4 Domination of $g(x^k) - g(x^*)$

By again using the strong convexity of g (as in (12)), we have

$$\begin{aligned}
& \mathcal{L}(\theta^k) - \gamma\big[g(x^k) - g(x^*)\big] \\
\geqslant\ & (2h-\gamma)\big[g(x^k) - g(x^*)\big] + c\|x^k - x^*\|^2 \\
& + (y^k - y^*)^\top \left[a_2(I - \frac{1}{n}ee^\top) + \frac{1}{n}ee^\top(na_1 + a_2 + n\mu d^2)\right](y^k - y^*) \\
& + (y^k - y^*)^\top e\left[2dg'(x^k) + 2b(x^k - x^*)\right] \\
\geqslant\ & (2h-\gamma)\big[g(x^k) - g(x^*)\big] + c\|x^k - x^*\|^2 \\
& - \frac{1}{4}\frac{n}{na_1 + a_2 + n\mu d^2}\|2dg'(x^k) + 2b(x^k - x^*)\|^2 \text{ by minimizing with respect to } y^k, \\
\geqslant\ & \|x^k - x^*\|^2\left[\frac{L}{2}(2h-\gamma) + c - \frac{n}{na_1 + a_2 + n\mu d^2}\big(dL + b\big)^2\right],
\end{aligned}$$

using the smoothness of g and the assumption that $2h - \gamma \leqslant 0$.

Thus, in order for $\mathcal{L}(\theta^k)$ to dominate $g(x^k) - g(x^*)$ we require the additional constraints

$$\frac{L}{2}(2h-\gamma) + c - \frac{n}{na_1 + a_2 + n\mu d^2}\big(dL + b\big)^2 \geqslant 0$$

$$na_1 + a_2 + n\mu d^2 \geqslant 0$$

$$2h - \gamma \leqslant 0.$$

B.5 Finding constants

Given the algorithm and result parameters (α, γ, δ), we have the following constraints:

$$\begin{aligned}
h &\geqslant 0 && \text{(Constraint 1)} \\
2h - \gamma &\leqslant 0 && \text{(Constraint 2)} \\
B_3 &> 0 && \text{(Constraint 3)} \\
B_4 &> 0 && \text{(Constraint 4)} \\
na_1 + a_2 + n\mu d^2 &\geqslant 0 && \text{(Constraint 5)} \\
\frac{L}{2}(2h-\gamma) + c - \frac{n}{na_1 + a_2 + n\mu d^2}\big(dL + b\big)^2 &\geqslant 0 && \text{(Constraint 6)} \\
C_2 &\leqslant 0 && \text{(Constraint 7)} \\
C_1 + C_3 &\leqslant 0 && \text{(Constraint 8)} \\
C_0 + \mu(C_1 + C_3) + \mu^2 C_2 &\leqslant 0 && \text{(Constraint 9)}
\end{aligned}$$

We also require $C_1 \leqslant 0$, but this will follow from Constraint 8 since we will have $C_3 \geqslant 0$. All of the constraints above are convex in a_1, a_2, b, c, d, h. Thus, given candidate values for the remaining values, the feasibility of these constraints may be checked using a numerical toolbox (as a second-order cone program). However, these parameters should be functions of n and should be valid for all μ. Thus, given a sampling of values of μ and n, representing these parameters as polynomials in $1/n$, the candidate functions may be found through a second-order cone programs.

By experimenting with this strategy, we have come up with the following values for the constants:

$$\begin{aligned}
a_1 &= \frac{1}{32nL}\left(1-\frac{1}{2n}\right) \\
a_2 &= \frac{1}{16nL}\left(1-\frac{1}{2n}\right) \\
b &= -\frac{1}{4n}\left(1-\frac{1}{n}\right) \\
c &= \frac{4L}{n} \\
h &= \frac{1}{2}-\frac{1}{n} \\
d &= \frac{\alpha}{n} \\
\alpha &= \frac{1}{16L} \\
\delta &= \min\left(\frac{1}{8n}, \frac{\mu}{16L}\right) \\
\gamma &= 1 \\
C_3 &= \frac{1}{32n}.
\end{aligned}$$

We will assume $n > 1$, since the result for $n = 1$ follows because SAG is equivalent to gradient descent in this case. Under this assumption, we see that Constraints 1 and 2 are satisfied by the above parameterizaiton. Below we verify that Constraints 3-9 are also satisfied using symbolic computations in Matlab (the code used in this section is available on the first author's webpage). Specifically, we parameterize the constraints as polynomials and then verify that the polynomials are positive over an appropriate interval using the function below (which simply checks that no roots of the polynomial P lie in the interval (x_1, x_2), and that P is positive at the mid-point of the interval).

```
function [check] = isPositive(P,x1,x2)
% Returns 1 if univariate polynomial P has no
% no real roots in the interval [x1,x2], and
% is positive at the mid-point.
%
% This is a sufficient condition for P to
% be positive on the interval [x1,x2].
p = sym2poly(P);
r = roots(p);
check = polyval(p,(x2+x1)/2)>0 & all( abs(imag(r))>1e-12 | (r <= x1) |(r >= x2) );
```

B.6 Verifying the result

To verify the result under these constants, we consider whether $\delta = 1/8n$ or $\mu/16L$. In B_4, we discard the term of the form $(1-\delta)\mu h d^2$, which does not impact the validity of our result. Moreover, without loss of generality, we may assume $L = 1$ and $\mu \in [0, 1]$.

B.6.1 Well-conditioned problems ($\mu \geqslant 2/n$)

In this situation, it suffices to show the result for $\mu = 2/n$ since, (a) if a function is μ-strongly convex, then it is μ'-strongly convex for all smaller μ', and (b) the final inequality does not involve μ. All constraints (when properly multiplied where appropriate by $B_3 B_4$) can be written as rational

functions of $x = 1/n$:

$$B_3 \qquad \text{(Constraint 3)}$$
$$B_4 \qquad \text{(Constraint 4)}$$
$$na_1 + a_2 + n\mu d^2 \qquad \text{(Constraint 5)}$$
$$\big[\frac{L}{2}(2h-\gamma)+c\big](na_1+a_2+n\mu d^2) - n\big(dL+b\big)^2 \qquad \text{(Constraint 6)}$$
$$-C_2B_3B_4 \qquad \text{(Constraint 7)}$$
$$-(C_1+C_3)B_3B_4 \qquad \text{(Constraint 8)}$$
$$-B_4B_3(C_0+(C_1+C_3)\mu+C_2\mu^2) \qquad \text{(Constraint 9)}$$

We only need to check the positivity of these polynomials in x. We do this using the function above, via the script below. Note that we also have $B_3 > 0$ and $B_4 > 0$ for $n > 1$.

```
syms n mu delta d  h c b a1 a2 x
syms B0 B1 B2 B3 B4 B5 B6 B7 B8 B9
syms C0_B4 C13_B3B4 C2_B3B4 C3
syms tocheck_1 tocheck_2 tocheck_3 tocheck_4 tocheck_5 tocheck_6 tocheck_7

% parameters of the Lyapunov function
alpha = 1/16;
n = 1/x;
mu = 2/n;
delta = 1/8/n;
d = alpha/n;
gamma = 1;
h = 1/2 - 1/n ;
c = 4/n;
b = -1/4* ( 1/n - 1/n/n) ;
a1 = (1 - 1/n/2)/n/32;
a2 = (1 - 1/n/2)/n/16;
C3 = 1/32/n;

% compute all parameters
B0 = 2 * delta * h;
B1 = 2 * ( b - alpha/n * c);
B2 = 2 * ( alpha/n - d)*h;
B3 = - ( (1-2/n)*a2 + 1/n*(a1+a2 -2*alpha/n*b + alpha*alpha/n/n*c )-(1-delta)*a2);
% modified value of B4 below
B4 = - n * ( (1-2/n)*(a1-2*alpha/n*b+alpha*alpha/n/n*c) - (1-delta)*a1 ) + B3;
B5 = 2* (  ( delta-1/n)*b - alpha/n*(1-1/n)*c);
B6 = -2/n * ( a1 - 2*alpha/n*b + alpha*alpha/n/n*c );
B7 = 2 * ( a1 - 2*alpha/n*b + alpha*alpha/n/n*c ) + 2 * ( - h*(1-delta)*d );
B8 = 1/n * ( a1+a2- 2*alpha/n*b + alpha*alpha/n/n*c );
B9 = c * delta;

C0_B4 = - B0*mu/2*B4 + B9*B4 +n/4*B5*B5   ;
C13_B3B4 = B0*B3*B4+B1*B3*B4 + n/4*B6*B6*B4+n/2*B5*(B6+B7)*B3+n*B8*B3*B4 + C3*B3*B4;
C2_B3B4 = -B2*B3*B4-n/4*B6*B6*B4 + n/4*(B6+B7)*(B6+B7)*B3;

isPositive(B3,0,1/2) % Constraint 3
isPositive(B4,0,1/2) % Constraint 4
isPositive(n*a1+a2+n*mu*d*d,0,1/2) % Constraint 5
isPositive((1/2*(2*h-gamma) + c)*(n*a1+a2+n*mu*d*d) - (d+b)*(d+b)*n,0,1/2) % Constraint 6
isPositive(-C2_B3B4,0,1/2) % Constraint 7
isPositive(-C13_B3B4,0,1/2) % Constraint 8
isPositive(-(C0_B4*B3+C13_B3B4*mu+C2_B3B4*mu*mu),0,1/2) % Constraint 9
```

B.6.2 Ill-conditioned problems ($\mu \leqslant 2/n$)

We consider the variables $x = 1/n \in [0, 1/2]$ and $y = n\mu/2 \in [0, 1]$, so that $\mu = 2y/n$. We may express all quantities using x and y. The difficulty here is that we have two variables. We first

check the dependency in terms of y of the expressions B_3, B_4, and $B_6 + B_7$. These are univariate polynomials with an affine dependence on y, so their positivity can be checked by checking positivity for $y = 1$ or $y = 0$. Again making use of symbolic computation, we can deduce that

$$\begin{aligned} B_3 \quad & \text{is non-negative and decreasing in } y, && \text{(Constraint 3)} \\ B_4 \quad & \text{is non-negative and decreasing in } y, && \text{(Constraint 4)} \\ B_6 + B_7 \quad & \text{is non-negative and increasing in } y, \end{aligned}$$

We include below the additional commands to verify these properties:

```
syms y

% Re-define mu and delta
mu = 2/n*y;
delta = mu/16;

% Re-define B3, B4, and B7 (they depend on delta)
B3 = - ( (1-2/n)*a2 + 1/n*(a1+a2 -2*alpha/n*b + alpha*alpha/n/n*c )-(1-delta)*a2);
B4 = - n * ( (1-2/n)*(a1-2*alpha/n*b+alpha*alpha/n/n*c) - (1-delta)*a1 ) + B3;
B7 = 2 * ( a1 - 2*alpha/n*b + alpha*alpha/n/n*c ) + 2 * ( - h*(1-delta)*d );

isPositive(subs(B3,y,0) - subs(B3,y,1),0,1/2) % B3 is decreasing
isPositive(subs(B3,y,1),0,1/2) % B3 is positive when y==1 (Constraint 3)
isPositive(subs(B4,y,0) - subs(B3,y,1),0,1/2) % B4 is decreasing
isPositive(subs(B4,y,1),0,1/2) % B4 is positive at y=1 (Constraint 4)
isPositive(simplify(subs(B6+B7,y,1) - subs(B6+B7,y,0)),0,1/2) % B6+B7 is increasing
isPositive(subs(B6+B7,y,0),0,1/2) % B6+B7 is positive when y==0
```

Given the monotonicity of the bounds in B_3 and B_4, we only need to check our results for the smaller values of B_3 and B_4, i.e., for $y = 1$. Similarly, because of the monotonicity in the term $(B_6 + B_7)^2$, we may replace $(B_6 + B_7)^2$ by $(B_6 + B_7)$ times its upper-bound (i.e., its value at $y = 1$). Also note that B_5 is divisble by y, and we have $B_3 > 0$ and $B_4 > 0$ for $n > 1$. Using this, we can show that Constraints 5-9 are satisfied by checking the positivity of the following polynomials:

$$\begin{aligned} & na_1 + a_2 + n\mu d^2 && \text{(Constraint 5)} \\ & \left[\frac{L}{2}(2h - \gamma) + c\right](na_1 + a_2 + n\mu d^2) - n\big(dL + b\big)^2 && \text{(Constraint 6)} \\ & - C_2 B_3 B_4 && \text{(Constraint 7)} \\ & - (C_1 + C_3) B_3 B_4 && \text{(Constraint 8)} \\ & - B_4 B_3 (C_0/\mu + (C_1 + C_3) + C_2\mu) && \text{(Constraint 9)} \end{aligned}$$

Constraints 5-7 are affine in y and we thus only need to check positivity for $y = 0$ and $y = 1$.

```
deltamax = 1/8/n;

B0 = 2 * delta * h;
B3 = - ( (1-2/n)*a2 + 1/n*(a1+a2 -2*alpha/n*b + alpha*alpha/n/n*c )-(1-deltamax)*a2);
B4 = - n * ( (1-2/n)*(a1-2*alpha/n*b+alpha*alpha/n/n*c) - (1-deltamax)*a1 ) + B3;
B5 = 2* (  ( delta-1/n)*b - alpha/n*(1-1/n)*c);
B7max = 2 * ( a1 - 2*alpha/n*b + alpha*alpha/n/n*c ) + 2 * ( - h*(1-deltamax)*d );
B9 = c * delta;

C0_B4_mu = - B0/2*B4 + B9*B4/mu +n/4*B5*B5/mu;
C13_B3B4 = B0*B3*B4+B1*B3*B4 + n/4*B6*B6*B4+n/2*B5*(B6+B7)*B3+n*B8*B3*B4 + B3*B4*C3;
C2_B3B4 = -B2*B3*B4-n/4*B6*B6*B4 + n/4*(B6+B7)*(B6+B7max)*B3;

% Check expressions that are affine in y
for P = [n*a1+a2+n*mu*d*d, ... % Constraint 5
        (1/2*(2*h-gamma) + c)*(n*a1+a2+n*mu*d*d) - (d+b)*(d+b)*n,... % Constraint 6
        -C2_B3B4] % Constraint 7
    isPositive(subs( P , y , 0 ),0,1/2);
    isPositive(subs( P , y , 1 ),0,1/2);
end
```

The other two expressions (Constraints 8 and 9) are more complicated as there are second-order polynomials in y. If $n \geqslant 5$, they have positive second derivatives, negative derivatives at $y = 0$ and $y = 1$, and positive values at $y = 1$. They are thus positive, which is verified by the code below.

```
% Check second-order expressions in y
for P = [-C13_B3B4, ... % Constraint 8
        -(C0_B4_mu*B3+C13_B3B4+C2_B3B4*mu )] % Constraint 9
    temp = simplify(P);
    param0 =  subs( temp , y , 0 );
    param1 =  subs( diff(temp,y), y , 0 );
    param2 =  subs( diff(temp,y,2), y , 0 )/2;

    isPositive(param2,0,1/5) % check that second order is positive
    isPositive(-param1,0,1/5) % check that first order is negative
    isPositive(-param1 - 2*param2,0,1/5) % gradient at 1 is negative
    isPositive(subs( temp , y , 1 ),0,1/5) % simply check at 1
end
```

For the remaining scenarios where $n \in \{2, 3, 4\}$, we have a fixed x so we can check the positivity of the polynomials in y. A Matlab script that does the steps above in addition to checking these not-particularly-interesting cases is available from the first author's web page.

B.7 Convergence Rate

We have that

$$g(x^k) - g(x^*) \leqslant \mathcal{L}(\theta^k),$$

and that

$$\mathbb{E}(\mathcal{L}(\theta^k)|\mathcal{F}_{k-1}) - (1-\delta)\mathcal{L}(\theta^{k-1}) \leqslant -\frac{1}{32n}(x^{k-1} - x^*)^\top g'(x^{k-1}) \leqslant 0.$$

In the strongly-convex case ($\delta > 0$), combining these gives us the linear convergence rate

$$\mathbb{E}(g(x^k)) - g(x^*) \leqslant (1-\delta)^k \mathcal{L}(\theta^0).$$

In the convex case ($\mu = 0$ and $\delta = 0$), we have by convexity that

$$-\frac{1}{32n}(x^{k-1} - x^*)^\top g'(x^{k-1}) \leqslant \frac{1}{32n}[g(x^*) - g(x^{k-1})],$$

We then have

$$\frac{1}{32n}[g(x^{k-1}) - g(x^*)] \leqslant L(\theta^{k-1}) - \mathbb{E}(L(\theta^k)|\mathcal{F}_{k-1}).$$

Summing this up to iteration k yields

$$\frac{1}{32n}\sum_{i=1}^{k}[\mathbb{E}(g(x^{i-1}))-g(x^*)] \leqslant \sum_{i=1}^{k}\mathbb{E}[L(\theta^{i-1})-L(\theta^i)] = L(\theta^0)-\mathbb{E}[L(\theta^k)] \leqslant L(\theta^0).$$

Finally, by Jensen's inequality note that

$$\mathbb{E}\left[g\left(\frac{1}{k}\sum_{i=0}^{k-1}x^i\right)\right] \leq \frac{1}{k}\sum_{i=0}^{k-1}\mathbb{E}[g(x^i)],$$

so with $\bar{x}^k = \frac{1}{k}\sum_{i=0}^{k-1}x^i$ we have

$$\mathbb{E}[g(\bar{x}^k)]-g(x^*) \leqslant \frac{32n}{k}L(\theta^0).$$

B.8 Intial values of Lyapunov function

To obtain the results of Theorem 1, all that remains is computing the initial value of the Lyapunov function for the two initializations.

B.8.1 Initialization with the zero vector

If we initialize with $y^0 = 0$ we have

$$\begin{aligned}\mathcal{L}(\theta^0) = 2h(g(x^0)-g(x^*)) &+ f'(x^*)^\top(a_1 ee^\top + a_2 I)f'(x^*)\\ &+ 2bf'(x^*)^\top e(x^0-x^*) + c(x^0-x^*)^\top(x^0-x^*).\end{aligned}$$

Plugging in the values of our parameters,

$$\begin{aligned} h = \frac{1}{2}-\frac{1}{n}, \qquad a_1 &= \tfrac{1}{32nL}\left(1-\tfrac{1}{2n}\right), \qquad a_2 = \frac{1}{16nL}\left(1-\frac{1}{2n}\right),\\ b &= -\tfrac{1}{4n}\left(1-\tfrac{1}{n}\right), \qquad c = \frac{4L}{n},\end{aligned}$$

and using $\sigma^2 = \frac{1}{n}\sum_i \|f_i'(x^*)\|^2$ we obtain (noting that $e^\top f'(x^*) = 0$)

$$\begin{aligned}\mathcal{L}(\theta^0) &= \left(1-\frac{2}{n}\right)(g(x^0)-g(x^*)) + \left(1-\frac{1}{2n}\right)f'(x^*)^\top(\frac{1}{32nL}ee^\top + \frac{1}{16nL}I)f'(x^*)\\ &\qquad - \frac{1}{2n}\left(1-\frac{1}{n}\right)f'(x^*)^\top e(x^0-x^*) + \frac{4L}{n}(x^0-x^*)^\top(x^0-x^*)\\ &\leqslant g(x^0)-g(x^*) + \frac{\sigma^2}{16L} + \frac{4L}{n}\|x^0-x^*\|^2\end{aligned}$$

B.8.2 Initialization with average gradient

If we initialize with $y_i^0 = y_i^0 = f_i'(x^0) - (1/n)\sum_i f_i'(x^0)$ we have, by noting that we still have $(y^0)^\top e = 0$,

$$\mathcal{L}(\theta^0) = \left(1-\frac{2}{n}\right)(g(x^0)-g(x^*)) + \frac{1}{16nL}\left(1-\frac{2}{n}\right)\|y^0-f'(x^*)\|^2 + \frac{4L}{n}\|x^0-x^*\|^2.$$

By observing that $y^0 = f'(x^0) - eg'(x^0)$ and using [Nesterov, 2004, Equations 2.17], we can bound the norm in the second term as follows:

$$\begin{aligned}
\|y^0 - f'(x^*)\|^2 &= \|(f'(x^0) - f'(x^*)) - e(g'(x^0) - g'(x^*)\|^2 \\
&\leqslant 2\|f'(x^0) - f'(x^*)\|^2 + 2\|e(g'(x^0) - g'(x^*))\|^2 \\
&= 2\sum_{i=1}^{n} \|f_i'(x^0) - f_i'(x^*)\|^2 + 2n\|g'(x^0) - g'(x^*)\|^2 \\
&\leqslant 4L\sum_{i=1}^{n}[f_i(x^0) - f_i(x^*) - f_i'(x^*)^\top(x_0 - x^*)] + 4nL(g(x^0) - g(x^*)) \\
&= 8nL(g(x^0) - g(x^*)).
\end{aligned}$$

Using this in the Lyapunov function we obtain

$$L(\theta^0) \leqslant \frac{3}{2}(g(x^0) - g(x^*)) + \frac{4L}{n}\|x^0 - x^*\|^2.$$

References

A. Agarwal and L. Bottou. A lower bound for the optimization of finite sums. *arXiv preprint*, 2014.

A. Agarwal, P. L. Bartlett, P. Ravikumar, and M. J. Wainwright. Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization. *IEEE Transactions on Information Theory*, 58(5), 2012.

F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing Systems*, 2011.

F. Bach and E. Moulines. Non-strongly-convex smooth stochastic approximation with convergence rate $o(1/n)$. *arXiv preprint*, 2013.

D. P. Bertsekas. A new class of incremental gradient methods for least squares problems. *SIAM Journal on Optimization*, 7(4):913–926, 1997.

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

Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

L. Bottou and Y. LeCun. Large scale online learning. *Advances in Neural Information Processing Systems*, 2003.

P. Carbonetto. *New probabilistic inference algorithms that harness the strengths of variational and Monte Carlo methods*. PhD thesis, Univ. of British Columbia, May 2009.

R. Caruana, T. Joachims, and L. Backstrom. KDD-cup 2004: results and analysis. *ACM SIGKDD Newsletter*, 6(2):95–108, 2004.

A.L. Cauchy. 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.

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

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

G. V. Cormack and T. R. Lynam. Spam corpus creation for TREC. In *Proc. 2nd Conference on Email and Anti-Spam*, 2005. `http://plg.uwaterloo.ca/~gvcormac/treccorpus/`.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 2014a.

Aaron J Defazio, ANUEDU AU, Tibério S Caetano, NICTA COM AU, and Justin Domke. Finito: A faster, permutable incremental gradient method for big data problems. *International Conference on Machine Learning*, 2014b.

B. Delyon and A. Juditsky. Accelerated stochastic approximation. *SIAM Journal on Optimization*, 3(4):868–881, 1993.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal of Scientific Computing*, 34(3):A1351–A1379, 2012.

S. Ghadimi and G. Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization. *Optimization Online*, July, 2010.

Pinghua Gong and Jieping Ye. Linear convergence of variance-reduced projected stochastic gradient without strong convexity. *arXiv preprint*, 2014.

I. Guyon. Sido: A phamacology dataset, 2008. URL `http://www.causality.inf.ethz.ch/data/SIDO.html`.

E. Hazan and S. Kale. Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. *Conference on Learning Theory*, 2011.

C. Hu, J.T. Kwok, and W. Pan. Accelerated gradient methods for stochastic optimization and online learning. *Advances in Neural Information Processing Systems*, 2009.

R. Johnson and T Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*, 2013.

S.S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6:341–361, 2005.

H. Kesten. Accelerated stochastic approximation. *Annals of Mathematical Statistics*, 29(1):41–59, 1958.

Jakub Konečnỳ and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv preprint*, 2013.

Jakub Konečnỳ, Zheng Qu, and Peter Richtárik. Semi-stochastic coordinate descent. *arXiv preprint*, 2014.

H. J. Kushner and G. Yin. *Stochastic approximation and recursive algorithms and applications*. Springer-Verlag, Second edition, 2003.

S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *International Conference on Machine Learning*, 2013.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. *Advances in Neural Information Processing Systems*, 2012.

D.D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Qihang Lin, Zhaosong Lu, and Lin Xiao. An accelerated proximal coordinate gradient method and its application to regularized empirical risk minimization. *arXiv preprint*, 2014.

J. Liu, J. Chen, and J. Ye. Large-scale sparse logistic regression. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

Zhi-Quan Luo and Paul Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

M. Mahdavi and R. Jin. Mixedgrad: An $o(1/t)$ convergence rate algorithm for stochastic smooth optimization. *Advances in Neural Information Processing Systems*, 2013.

Julien Mairal. Optimization with first-order surrogate functions. *International Conference on Machine Learning*, 2013.

Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *arXiv preprint*, 2014.

J. Martens. Deep learning via Hessian-free optimization. *International Conference on Machine Learning*, 2010.

A. Nedic and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic, 2000.

D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Advances in Neural Information Processing Systems*, 2014.

A. Nemirovski and D. B. Yudin. *Problem complexity and method efficiency in optimization*. Wiley, 1983.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady AN SSSR*, 269(3):543–547, 1983.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer, 2004.

Y. Nesterov. Gradient methods for minimizing composite objective function. *CORE Discussion Papers*, 2007.

Y. Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *CORE Discussion Paper*, 2010.

Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1): 127–152, 2005.

Atsushi Nitanda. Stochastic proximal gradient descent with acceleration techniques. *Advances in Neural Information Processing Systems*, 2014.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, second edition, 2006.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Zheng Qu, Peter Richtárik, and Tong Zhang. Randomized dual coordinate ascent with arbitrary sampling. *arXiv preprint arXiv:1411.5873*, 2014.

A. Rakhlin, O. Shamir, and K. Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. *International Conference on Machine Learning*, 2012.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Christian P Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2nd edition, 2004.

M. Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab. `https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`, 2005.

M. Schmidt and N. Le Roux. Fast convergence of stochastic gradient descent under a strong growth condition. *arXiv preprint*, 2013.

M. Schmidt, N. Le Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in Neural Information Processing Systems*, 2011.

M. Schmidt, R. Babanezhad, M.O. Ahemd, A. Clifton, and A. Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. *International Conference on Artificial Intelligence and Statistics*, 2015.

S. Shalev-Schwartz and T. Zhang. Proximal stochastic dual coordinate ascent. *arXiv preprint*, 2013a.

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

S. Shalev-Schwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *International Conference on Machine Learning*, 2014.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Shai Shalev-Shwartz and Tong Zhang. Accelerated mini-batch stochastic dual coordinate ascent. *Advances in Neural Information Processing Systems*, 2013.

Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. *International Conference on Machine Learning*, 2014.

M.V. Solodov. Incremental gradient algorithms with stepsizes bounded away from zero. *Computational Optimization and Applications*, 11(1):23–35, 1998.

N. Srebro and K. Sridharan. Theoretical basis for "more data less work"? *NIPS Workshop on Computataional Trade-offs in Statistical Learning*, 2011.

T. Strohmer and R. Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

P. Sunehag, J. Trumpf, SVN Vishwanathan, and N. Schraudolph. Variable metric stochastic approximation theory. *International Conference on Artificial Intelligence and Statistics*, 2009.

Taiji Suzuki. Stochastic dual coordinate ascent with alternating direction method of multipliers. *International Conference on Machine Learning*, 2014.

C. H. Teo, Q. Le, A. J. Smola, and S. V. N. Vishwanathan. A scalable modular convex solver for regularized risk minimization. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2007.

P. Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, 8(2):506–531, 1998.

Chong Wang, Xi Chen, Alex Smola, and Eric Xing. Variance reduction for stochastic gradient optimization. *Advances in Neural Information Processing Systems*, 2013.

L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(2):2057–2075, 2014.

Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. Linear convergence with condition number independent access of full gradients. *Advances in Neural Information Processing Systems*, 2013.

Yuchen Zhang and Lin Xiao. Stochastic primal-dual coordinate method for regularized empirical risk minimization. *arXiv preprint*, 2014.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling. *arXiv preprint arXiv:1401.2753*, 2014.

L.W. Zhong and J.T. Kwok. Fast stochastic alternating direction method of multipliers. *International Conference on Machine Learning*, 2014.