Fast Variance Reduction Method with Stochastic Batch Size

Xuanqing Liu [1] **Cho-Jui Hsieh** [1 2]

Abstract

In this paper we study a family of variance reduction methods with randomized batch size—at each step, the algorithm first randomly chooses the batch size and then selects a batch of samples to conduct a variance-reduced stochastic update. We give the linear convergence rate for this framework for composite functions, and show that the optimal strategy to achieve the optimal convergence rate per data access is to always choose batch size of 1, which is equivalent to the SAGA algorithm. However, due to the presence of cache/disk IO effect in computer architecture, the number of data access cannot reflect the running time because of 1) random memory access is much slower than sequential access, 2) when data is too big to fit into memory, disk seeking takes even longer time. After taking these into account, choosing batch size of 1 is no longer optimal, so we propose a new algorithm called SAGA++ and show how to calculate the optimal average batch size theoretically. Our algorithm outperforms SAGA and other existing batched and stochastic solvers on real datasets. In addition, we also conduct a precise analysis to compare different update rules for variance reduction methods, showing that SAGA++ converges faster than SVRG in theory.

1. Introduction

In this paper, we consider the following finite-sum composite optimization problem:

$$w^* = \arg\min_{w\in\mathbb{R}^p} \Big\{ F(w) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i(w) + g(w) \Big\}. \quad (1)$$

Here we assume each $f_i(w)$ is a μ-strongly convex, L-smooth function, the regularization term $g(w)$ is convex but not necessarily differentiable. In machine learning applications, n is the number of training samples, each f_i is the loss function such as logistic loss or ℓ_2 loss, and $g(w)$ is the regularization term which can be non-smooth (e.g., ℓ_1 regularization). For large data, SGD is preferred over gradient descent and has been widely used in large-scale applications. However, since the variance of stochastic gradient will not go to zero even when $w = w^*$ (the optimal solution), SGD has to gradually shrink the step size to guarantee convergence, at the cost of suboptimal rate. To speed up the convergence, there is a recent line of research on developing new algorithms with linear convergence rate using variance reduction techniques, the representative work includes SAG (Hofmann et al., 2015), SVRG (Johnson & Zhang, 2013), SAGA (Defazio et al., 2014), S2GD (Konecnỳ & Richtárik, 2013) etc. Further, one can accelerate this framework via concepts similar to the Nesterov's momentum method (Lin et al., 2015; Allen-Zhu, 2017).

The motivation of this paper is to study the effect of batch size in variance reduction methods. The effect of batch size in SGD (without variance reduction) has been studied in the literature such as (Li et al., 2014; Bengio, 2012; Keskar et al., 2016). Assuming a subset of b samples is chosen for SGD at each step, the theoretical analysis in (Dekel et al., 2012) suggests that the error is at the order of $\mathcal{O}(1/\sqrt{bT} + 1/T)$ after T iterations, and this bound is later improved to $\mathcal{O}(1/\sqrt{bT})$ in (Li et al., 2014). When constraining on SVM-hinge loss, (Takác et al., 2013) also shows an order of $\mathcal{O}(\frac{n}{b} + \frac{\beta_b}{b}\cdot\frac{1}{\lambda\epsilon})$ iterations to get an ϵ-suboptimal solution. Since each iteration will take the time proportional to b, these bounds suggest that the acceleration of convergence exactly covers the overhead of each iteration. It is thus interesting to see whether the same conclusion also applies for variance reduction methods.

To answer this question, we study a family of variance reduction methods with randomized batch sizes. At each iteration, the algorithm first randomly selects the batch size and then chooses a batch of samples to conduct a variance reduced stochastic update. Our main findings and contributions can be listed as follows:

- We prove linear convergence rate for this family of

[1]Department of Computer Science, University of California, Davis, California, USA [2]Department of Statistic, University of California, Davis, California, USA. Correspondence to: Xuanqing Liu <xqliu@ucdavis.edu>, Cho-Jui Hsieh <chohsieh@ucdavis.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

stochastic batched variance reduction algorithms. Our result covers composite minimization problems with non-smooth regularizations, and any distribution of batch sizes.

- Interestingly, with this unified analysis, we theoretically show that the convergence rate can be maximized if the algorithm always chooses batch size of 1. Therefore, increasing batch size does not help in terms of the number of data access.
- However, the number of data access does not precisely reflect the actual running time due to the memory hierarchy and cache/disk IO effect in computer architectures—accessing a continuous block of memory is faster than accessing disjoint ones, and disk seeking costs even more. After taking these into account, we propose the SAGA++ algorithm, and show how to calculate the optimal average batch size in practice. Our algorithm outperforms existing algorithms in terms of running time.
- In addition, we also develop a more precise analysis for comparing the convergence rates of variance reduction methods, and develop an algorithm to universally accelerate the stochastic methods for solving ℓ_1-regularized problems by lazy updates. Which rediscovers (Konečnỳ et al., 2016) independently.

Related Work We will discuss the related variance reduction methods in next section. Here we describe some other related work on stochastic optimization.

Stochastic optimization has become popular due to their vast and far reaching applications in large-scale machine learning, and this is also one of our main focus in this paper. Among them, stochastic gradient descent has been widely used, and its variants (Duchi et al., 2011; Kingma & Ba, 2014) are popular for training deep neural networks. There are also other examples, such as stochastic coordinate descent (Nesterov, 2012) and stochastic dual coordinate descent (Shalev-Shwartz & Zhang, 2013). At each iteration, SGD selects one sample to conduct the update, but its gradient often contains huge noise. To reduce the noise or variance, mini-batch SGD has been intensively studied in the literature, including (Li et al., 2014), and the recent work on big-batch SGD (De et al., 2016). Some theoretical results have been discussed in our introduction (Li et al., 2014; Dekel et al., 2012).

Although some recent works have discussed about mini-batch variance reduction algorithms (Hofmann et al., 2015; Harikandeh et al., 2015), there is no clear conclusion on whether increasing the batch size helps the convergen speed. Ideally the convergence rate $1-\rho$ should linearly depend on the batch size: $\rho \propto b$; if that is the case, simply by calculating the batched gradient in parallel we will see linear speed up. (Hofmann et al., 2015) suggests $\rho \approx b/n$ in big data regime and ρ is independent of b in ill-conditioned case, this can be regarded as an asymptotic situation of our result, which claims that ρ is a increasing function of b, but a larger batch size is less useful when the Hessian is ill conditioned. However, with a more precise bound in terms of b, we are able to show that $b = 1$ is always optimal in terms of number of data access.

As to the sampling techniques, the random sampling of batch size is seen in (Richtárik & Takáč, 2016) where the authors considered about the partially separable functions and apply block coordinate descent by randomly generate a set of blocks with arbitrary size. Similar idea is later exploited in (Qu et al., 2015; Csiba & Richtárik, 2015). Our idea differs from these previous works in that we put computer architecture effects into account when deciding whether we should choose full gradient or stochastic gradient to update parameters.

2. Framework: Variance Reduction with Stochastic Batch Size

Our proposed framework is shown in Algorithm 1: at each iteration, the algorithm first randomly chooses the batch size, ranging from 1 to n, and then samples a batch accordingly. We use $\mathcal{B}$ as a random set to denote the mini-batch chosen at each step, and its batch size, denoted as $|\mathcal{B}|$, is a random variable. Denote $f_i'(\phi_i^t)$ as the previous gradient evaluated on sample x_i and w^t is the iterate at time t. The update rule is given by:

$$w^{t+1} = \mathsf{Prox}_{\gamma g(\cdot)}\big(w^t - \gamma G(w^t)\big), \tag{2}$$

where γ is the step size and $G(w^t)$ is the unbiased gradient estimator:

$$G(w^t) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} f_i'(w^t) \underbrace{- \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} f_i'(\phi_i^t) + \bar{u}}_{\text{control variate}}, \tag{3}$$

where $\bar{u} = \frac{1}{n}\sum_{i=1}^n f_i'(\phi_i^t)$ is stored and maintained in memory. Similar to SAGA, in general the algorithm needs to store all the vectors $f_i'(\phi_i^t)$ in memory, but for many commonly used cases it only needs to store a scalar for each sample index i. For example, in GLM problems where $f_i(w) = \ell_i(x_i^\intercal w)$, since $f_i'(w) = x_i\ell_i'(x_i^\intercal w)$ we only need to store a scalar $\ell_i'(x_i^\intercal \phi_i^t)$ for each $i \in \{1, 2, \ldots, n\}$.

Algorithm 1 is very general and can include most of the existing variance reduction methods because our algorithm does not put any restriction on the choice of batch size, which can be either random or fixed. We discuss the connections between this framework and others:

- When the batch size is n with probability 1, Algorithm 1 will compute the full gradient at each iteration, which is equivalent to gradient descent.

Algorithm 1 Variance Reduction Method with Stochastic Batch Size

Input: training samples $\{(x_i, y_i)\}_{i=1}^n$, initial guess w_0
Output: $w^* = \arg\min_w F(w)$
$w = w_0$;
for `iter`$= 0$ **to** `MAX_ITER` **do**
  Choose a batch size $1 \leq b \leq n$ randomly based on some distribution;
  Sample a batch $\mathcal{B} \subseteq \{1, 2, \ldots, n\}$, $|\mathcal{B}| = b$;
  Calculate variance reduced gradient vector by (3);
  Apply update according to (2);
  Update gradient memory: $f_i'(\phi_i^{t+1}) \leftarrow f_i'(w^t)$, $\forall i \in \mathcal{B}$;
  Update $\bar{u} \leftarrow \bar{u} + \frac{1}{n}\sum_{i\in\mathcal{B}} f_i'(w_t) - \frac{1}{n}\sum_{i\in\mathcal{B}} f_i'(\phi_i^t)$
end for
Return $w^* = w$;

- When the batch size is always 1, the algorithm is equivalent to SAGA (Defazio et al., 2014). At each step, SAGA uniformly chooses one sample from $\{1, 2, \ldots, n\}$ and then update the iterates by the same variance reduced gradient defined in (3).
- SVRG (Johnson & Zhang, 2013): This method adopts two layers of iterations. In each outer iteration SVRG calculates full gradient (also called gradient snapshot), and in each inner iteration it chooses one sample to update. SVRG does not update the gradient snapshot inside the inner iteration, so strictly speaking it cannot fit into our framework. However our algorithm, SAGA++, based partly on SVRG adopts a better update rule which will be discussed later.
- S2GD, mS2GD: They are variants of SVRG when the number of inner iterations m follows a probability distribution: $m \sim (1-\nu\gamma)^{M-m}/\beta$, $m = 1, 2, \ldots, M$ where ν is the lower bound of strongly convex factor μ, β is a normalizing factor and γ is step size. (Konečnỳ et al., 2016) extends S2GD to mini-batched version.

3. Theoretical Analysis and New Algorithms

We discuss our theoretical results and new insights in this section. First, we prove the linear convergence rate of Algorithm 1 in Section 3.1, and then in Section 3.2 we will take the cache/disk IO effect into consideration to derive the new algorithm SAGA++. We show the SAGA-style update used in this paper is more efficient than SVRG-style update in Section 3.3 and then discuss a new technique to conduct lazy update for ℓ_1 regularization in our Algorithm 1. We left the proof in appendix.

3.1. Convergence rate analysis

We assume the objective function is μ-strongly convex and L-Lipschitz smooth, and $\kappa = L/\mu$ is the condition number. We will use the following useful bounds in our analysis:

$$f(y) \geq f(x) + f'(x)(y-x) + \mu/2\|y-x\|^2 \tag{4a}$$
$$f(y) \leq f(x) + f'(x)(y-x) + L/2\|y-x\|^2. \tag{4b}$$

Hereafter we use $\|\cdot\|$ to denote ℓ_2 norm unless stated explicitly. To simplify notation, we define $f_i^\delta(w) = f_i(w) - f_i(w^*) - f_i'(w^*)(w-w^*)$ and $f^\delta(w) = \frac{1}{n}\sum_{i=1}^n f_i^\delta(w)$ as the Bregman divergence between w and w^*, and define $\bar{H}_t = \frac{1}{n}\sum_{i=1}^n f_i^\delta(\phi_i^t)$ to represent the averaged Bregman divergence between w^* and the snapshots ϕ_i^t.

To show the convergence of Algorithm 1, we first calculate the expected change of $\bar{H}_t$ after each update:

Lemma 1 *For the update rule (2) and $\bar{H}_t$ defined above, we have $\mathbb{E}\bar{H}_{t+1} = \frac{n-\mathbb{E}|\mathcal{B}|}{n}\bar{H}_t + \frac{\mathbb{E}|\mathcal{B}|}{n} f^\delta(w^t)$.*

Note that unlike (Hofmann et al., 2015) where $|\mathcal{B}|$ is deterministic, here we generalize their result to allow random batch size. Similarly, the progress of $\|w^t - w^*\|$ can be bounded by:

Lemma 2 *Define iteration progress by the distance to the optimal solution w^*, then:*

$$\begin{aligned}\|w^t - w^*\|^2 - \mathbb{E}\|w^{t+1} - w^*\|^2 \geq \gamma\mu\|w^t - w^*\|^2 \\ -((1+\beta)\gamma^2 - \gamma/L)\mathbb{E}\|f_i'(w^t) - f_i'(w^*)\|^2 \\ +\gamma^2\beta\|f'(w^t) - f'(w^*)\|^2 + \frac{2\gamma(L-\mu)}{L} f^\delta(w^t) \\ -2(1+\beta^{-1})L\gamma^2\bar{H}_t,\end{aligned} \tag{5}$$

where $\beta > 0$ is an arbitrary constant.

Combining Lemma 1 with 2, we can build a contraction on Lyapunov function as follows.

Theorem 3 *Define Lyapunov function as $\mathcal{L}_t = c\bar{H}_t + \|w^t - w^*\|^2$, c is a predefined constant, then we have $\mathbb{E}\mathcal{L}_{t+1} \leq (1-\rho)\mathcal{L}_t$ where $\rho = \min(\frac{\mathbb{E}|\mathcal{B}|}{n} - \frac{2(1+\beta^{-1})L\gamma^2}{c}, \gamma\mu)$ if the step size γ satisfies the following conditions:*

Upper bound :

$$\gamma \leq \min\Big(\frac{1}{(1+\beta)L}, \sqrt{\frac{c\mathbb{E}|\mathcal{B}|}{2(1+\beta^{-1})nL}}\Big). \tag{6}$$

Lower bound :

$$2\mu\beta\gamma^2 + \frac{2(L-\mu)}{L}\gamma - \frac{c\mathbb{E}|\mathcal{B}|}{n} \geq 0.$$

Recall c, β are predefined constants.

However, the result above is too complex to interpret. To get a better understanding of how the averaged batch size $\mathbb{E}|\mathcal{B}|$, step size γ and contraction factor ρ are related to each other, we simplify the result along different directions.

Proposition 1 *(**Adaptive step size**) In this case we want the step size γ to be independent on strong convexity μ. To this end, we set $\beta = 2$, $c = \frac{n}{3L\mathbb{E}|\mathcal{B}|}$, so that $\gamma = \frac{1}{3L}$.*

This result is the same with the adaptive step size of SAGA, which trade simplicity with tightness (the step size and convergence rate is independent on $\mathbb{E}|\mathcal{B}|$ so we can not see the benefit of larger batch). To develop a more informative result, we resort to the following proposition:

Proposition 2 *($\mathbb{E}|\mathcal{B}|$**-dependent step size**) If we set step size to $\gamma = \frac{c}{8\kappa}\left(\sqrt{1+\frac{16\kappa\mathbb{E}|\mathcal{B}|}{cn\mu}}-1\right)$ and $c = \frac{\tau n}{L\mathbb{E}|\mathcal{B}|}$, $\tau \in (0,1)$ is a constant, our algorithm converges linearly with a contraction factor $1-\rho$, i.e. $\|w^t - w^*\|^2 \leq (1-\rho)^t\left[\|w^0 - w^*\|^2 + c\bar{H}_0\right]$ and $\rho = \gamma\mu$.*

The selection of step size in Proposition 2 is optimal in terms of maximizing the convergence rate, as proven in appendix. Admittedly, after developing these results, the convergence rate and step size are loose after many inequalities, so these bounds should be regarded as the worst case situation. Even so, as a quick verification, we can show that our result matches the bounds of gradient descent and SAGA in the following extreme cases:

- Gradient descent: when setting $\mathbb{P}(|\mathcal{B}| = n) = 1$, then $\rho_{\text{GD}} = \frac{\tau}{8\kappa^2}(\sqrt{16\kappa^2/\tau+1}-1) \approx \frac{\sqrt{\tau}}{2\kappa}$ this gives the same order as the standard rate of gradient descent ($\frac{2\mu}{L+\mu}$).
- SAGA: $\mathbb{P}(|\mathcal{B}| = 1) = 1$, for the ill-conditioned case where κ is comparable to n, $\rho = \frac{1}{\kappa}\cdot\text{constant} \approx O(\frac{1}{\kappa})$. While in the well-conditioned case, $\kappa \ll n$, we have $\rho = \frac{\tau n}{8\kappa^2}(\sqrt{1+\frac{16\kappa^2}{\tau n^2}}-1) \approx O(\frac{1}{n})$. These rates match the results in the original SAGA paper (Defazio et al., 2014).

Note that the step size, unlike SGD, is always bounded away from zero (we do not need to decrease the step size in each epoch). This can be seen from the fact that $\gamma = \frac{c}{8\kappa}\left(\sqrt{1+\frac{16\kappa\mathbb{E}|\mathcal{B}|}{cn\mu}}-1\right) \geq 0$.

Next we try to find out the optimal averaged batch size $\mathbb{E}|\mathcal{B}|$ in order to achieve the optimal convergence rate in terms of *number of data access*. Based on Proposition 2, to return an ϵ-accurate solution we need $\mathcal{O}(\frac{\log\epsilon}{\log(1-\rho)})$ iterations, which implies $\mathcal{O}(\frac{\log\epsilon}{\log(1-\rho)}\mathbb{E}|\mathcal{B}|)$ epochs if the averaged batch size is $\mathbb{E}|\mathcal{B}|$. We then derive the following corollary to show that simply increasing the batch size will slow down the convergence rate per data access:

Corollary 1 *(**Theoretically optimal batch size**) Since the effective number of data access per iteration is proportional to the averaged batch size $\mathbb{E}|\mathcal{B}|$, the optimal batch size should maximize the decrement of function value with fixed gradient calculation, which can be formulated as follows:*

$$\begin{aligned}\mathbb{E}|\mathcal{B}|^* &= \arg\min_{\mathbb{E}|\mathcal{B}|} \frac{\log\epsilon}{\log(1-\rho)}\mathbb{E}|\mathcal{B}| \approx \arg\min_{\mathbb{E}|\mathcal{B}|} \frac{\mathbb{E}|\mathcal{B}|}{\rho} \\ &= \arg\min_{\mathbb{E}|\mathcal{B}|} \frac{\mathbb{E}|\mathcal{B}|^2}{\sqrt{1+\frac{16\kappa^2}{\tau n^2}\mathbb{E}|\mathcal{B}|^2}-1}.\end{aligned} \tag{7}$$

By taking derivative it is easy to see that the function is monotone increasing with $\mathbb{E}|\mathcal{B}|$ (we leave it in appendix). So theoretically $\mathbb{E}|\mathcal{B}| = 1$ (which is SAGA method) is optimal.

3.2. SAGA++: Optimal batch sizes when taking cache/disk IO effect into consideration

According to Corollary 1, one should always choose $|\mathcal{B}| = 1$ in order to minimize the number of data accesses. However, small number of data access may not necessarily lead to short running time in practice—in modern computer architectures, "sequential accesses" of data stored in the memory is much faster than "random accesses", because accessing the memory wildly can result in frequent cache miss or disk seeking. Therefore, calculating the full gradient $f'(w)$ will take less time than calculating n random gradient components $f_i'(w)$ (see Table 1 for measurement result). This leads to a new variance reduction method with non-deterministic batch size selection strategy(SAGA++) that combines full gradient access and SAGA ($|\mathcal{B}| = 1$): at each step we choose $|\mathcal{B}| = n$ with probability p and $|\mathcal{B}| = 1$ with probability $1-p$. When $|\mathcal{B}| = n$, SAGA++ accesses the whole dataset and this can be relatively fast due to the sequential memory access pattern, while when $|\mathcal{B}| = 1$ it randomly accesses one sample. By changing p we can smoothly change the average batch size from 1 to n. Next we show how to take the cache/disk IO effect into consideration and derive the "optimal" p in theory, while in the experimental part we show that the optimal average batch size can be large, depending on the problem and data.

To derive the optimal average batch size that yields minimal running time, we assume the computer needs T_{seq} time to sequentially access n samples, and T_{rand} to randomly access the same number of samples. Therefore, when $|\mathcal{B}| = 1$, each update costs within T_{rand}/n time units. We call the ratio $T_{\text{seq}}/T_{\text{rand}}$ as the *cache effect ratio*.

Corollary 2 *(**Optimal batch size with cache/disk IO effect**) If $T_{seq}/T_{rand} = \eta$, $\eta < 1$, then the optimal average*

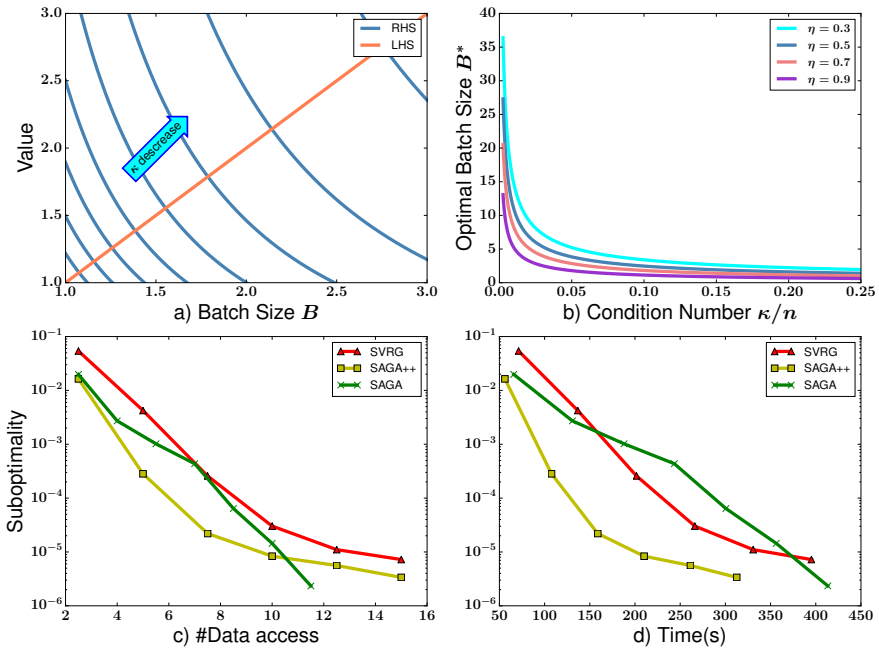

Figure 1. a) solution of (8) when cache/disk IO effect coefficient $\eta = 0.7$, the optimal batch size $\mathbb{E}\mathcal{B}^*$ is the intersection of two lines (marked as blue and orange), in this plot the condition number ranges from $0.025n$ to $0.5n$. b) To see the relation of $\mathbb{E}B^*$ and condition number κ more clearly, we solve (8) numerically, the optimal batch size drop rapidly when κ grows. At the same level of condition number, we should use a larger average batch when the cache/disk IO effect is strong (η small). c,d) Experiment on `avazu` dataset(cache/disk IO effect ratio $\eta = 0.46$), with respect to both data access(gradient calculation) and running time.

batch size will satisfy the following equations:

$$\mathbb{E}|\mathcal{B}^*| \approx (\frac{1}{\eta} - 1)\frac{\xi - 1}{2 - \xi}, \quad \alpha = \frac{4\kappa}{\sqrt{\tau}n},$$
$$\xi = \frac{\alpha^2 \mathbb{E}|\mathcal{B}^*|^2}{1 + \alpha^2 \mathbb{E}|\mathcal{B}^*|^2 - \sqrt{1 + \alpha^2 \mathbb{E}|\mathcal{B}^*|^2}}. \tag{8}$$

Note that ξ is also determined by $\mathbb{E}\mathcal{B}^*$ which makes the closed form solution intractable. However, if we know condition number κ and cache effect ratio η, the optimal batch size can be computed numerically. To gain more insights, we plot $\frac{\xi-1}{2-\xi}$ as a function of $\alpha \cdot \mathbb{E}\mathcal{B}^*$ in Figure 1(a), which shows the connection between the best batch size $\mathbb{E}\mathcal{B}^*$ and κ: in the well-conditioned regime we can select a larger batch size and in the ill-conditioned case a smaller batch size is better. Furthermore, Figure 1(b) reveals the optimal average batch size changes with condition number and cache effect ratio η: Conceptually, if η is smaller (sequential accesses are much faster than random accesses), then we are expected to do the full gradient update more frequently.

Our algorithm looks similar to SVRG—sometimes do a full gradient update while other times select a single instance. However, we use a different book-keeping strategy—SVRG does not update the gradient snapshot and control variate (defined in (3)) in between the two outer iterations, while SAGA++ will keep updating them even when batch size $= 1$. Since we always keep the latest information, the convergence speed is always better than SVRG, and we leave the detailed discussion to Section 3.3.

3.3. One step analysis: comparing Algorithm 1 with SVRG-style update

By far, we have only discussed the convergence speed under SAGA-style framework. In this section, we analyze why Algorithm 1 has faster convergence rate compared to the SVRG-style updates. This explains why SAGA++ (a special case of SVRG) is faster than SVRG, since they have the same data access pattern and differs in update rules. Here SVRG-style means we do not update the control variate in (3) before a new gradient snapshot is calculated, while in Algorithm 1 we store and update each gradient memorization $f_i'(\phi_i^t)$ as well as its summation once "fresher" gradient is available. Since the proposed framework in Algorithm 1 includes SAGA and SAGA++ as special cases, we call it "SAGA-style" update hereafter.

The main advantage of SVRG-style update is that it needs less memory, however, since many machine learning problems can be formulated as generalized linear model (GLM):

$f_i(w) = f(w^\intercal x_i)$, so the gradient $f_i'(w) = f'(w^\intercal x_i)x_i$ is purely determined by $w^\intercal x_i$, and SAGA-style update need only to store this scalar instead of the gradient vector for each sample. Therefore, for GLM problems the memory overhead of SAGA-style algorithm is simply an $\mathcal{O}(n)$ vector.

In terms of convergence rate, the following theory indicates that SAGA-style updates can better control the variance of gradient. First of all, we extend (3) to a more general variance reduced gradient defined by $G(w^t) = f_i'(w^t) - g_i$, where $g_i = \alpha_i - \frac{1}{n}\sum_{j=1}^n \alpha_i$ can be any zero-mean control variate. The update rules for SVRG, SAGA, and SAGA++ can be written as $\alpha_i = f_i'(w^\tau)$, where:

$$\begin{aligned} &\textbf{SVRG: } \tau = kT \\ &\textbf{SAGA: } 0 \le \tau \le t \\ &\textbf{SAGA++: } kT \le \tau \le t, \end{aligned} \tag{9}$$

where $f_i'(w^\tau) = f_i'(\phi_i^t)$ is stored in memory, T is the number of inner iterations inside each outer iteration and suppose the program have just finished the k-th outer iteration. We only consider the $|\mathcal{B}| = 1$ case since we want to focus on the control variate rather than batch size. For each i, by regarding τ as a random variable, we can calculate its probability distribution as follows:

$$\tau^{\text{SVRG}} = kT, \quad \tau^{\text{SAGA}} = \begin{cases} 0, & p = (1-\frac{1}{n})^t \\ 1, & p = \frac{1}{n}(1-\frac{1}{n})^{t-1} \\ \vdots & \\ t, & p = \frac{1}{n} \end{cases},$$

$$\tau^{\text{SAGA++}} = \begin{cases} kT, & p = (1-\frac{1}{n})^{t-kT} \\ kT+1, & p = \frac{1}{n}(1-\frac{1}{n})^{t-kT-1} \\ \vdots & \\ t, & p = \frac{1}{n}. \end{cases} \tag{10}$$

To see the difference of convergence rate between those update rules, we introduce the following lemmas:

Lemma 4 *If we use the distance $\|w^t - w^*\|^2$ as a metric to the sub-optimality, then we have:*

$$\begin{aligned} \mathbb{E}[\|w^{t+1} - w^*\|^2|\mathcal{F}_t] \le (1-\gamma\mu)\|w^t - w^*\|^2 \\ +(4L\gamma^2 - \frac{2\mu\gamma}{L} - 2\mu\gamma^2)f^\delta(w^t) \\ +2\gamma^2\mathbb{E}[\|\alpha_i - f_i'(w^*)\|^2|\mathcal{F}_t], \end{aligned} \tag{11}$$

where the expectation is taken over the choices of i ($\mathcal{F}_t$ is the σ-algebra at time t), $f^\delta(w) = f(w) - f(w^*) - f'(w^*)(w - w^*)$ is the Bregman divergence and we have $0 \le f^\delta(w) \le F(w) - F(w^*)$.

The first two terms in (11) is related to the distance between the current and optimal solution, only the last term involves the control variate α_i in different update rules, that is exactly what we are interested in, which can be further bounded by:

Lemma 5 *For an algorithm with $P(\tau = j) = p_j$, we can upper bound the gradient difference term:*

$$\mathbb{E}[\|\alpha_i - f_i'(w^*)\|^2|\mathcal{F}_0] \le 2L\sum_{j=1}^t p_j F^{sub}(w^j). \tag{12}$$

To see the change of $F^{\text{sub}}(w^t) = F(w^t) - F(w^*)$ with t, we claim that those variance reduction methods is expected to decrease in each step as long as γ is small(but keep to some constant):

Proposition 3 *For strongly convex function $f(w)$, define the update rule $w^{t+1} = w^t - \gamma G(w^t)$ (we ignore the regularization term for simplicity). If we want the function value to be a super-martingale, i.e. $\mathbb{E}[f(w^{t+1})|\mathcal{F}_t] \le f(w^t)$, then for SGD we require $\gamma \to 0$. But for variance reduction methods, since the variance of $G(w^t)$ goes to zero as fast as $f(w^t) - f^*$ (see appendix for details), a constant step size is enough.*

Finally we can compare the update rules listed in (3.3) by Proposition 3: Since the upper bound of distance improvement in (11) is determined by the variation of control variate $\mathbb{E}[\|\alpha_i - f_i'(w^*)\|^2|\mathcal{F}_0]$ and it is further upper bounded by (12), this can be seen as a weighted sum of expected function suboptimal $\mathbb{E}[F^{\text{sub}}|\mathcal{F}_0]$ and further from Corollary 3 we know F^{sub} is expected to decrease at each iteration, so we can conclude that more "weight" p_j should be put on smaller $\mathbb{E}[F^{\text{sub}}|\mathcal{F}_0]$, in another word, a good update rule should keep all the stochastic gradient active, rather than stale for too long. Therefore, by (10) we can observe that the distribution of p_j in SAGA++ is strictly better than both SAGA and SVRG, which indicates that SAGA++ has a faster convergence rate while keeping the same computational cost.

3.4. Lazy update for ℓ_1 regularization

For sparse datasets and $f_i(w) = f(w^\intercal x_i)$, the stochastic gradient $f_i'(w^\intercal x_i)x_i$ has the same indices of zero elements as x_i. However, the $\bar{u}$ vector in update rule (3) is a dense vector, which will lead to updating all the d variables. To reduce the time complexity back to $O(\texttt{nnz}(x_i))$ per step, a "lazy update" technique was discussed in (Schmidt et al.) for ℓ_2 regularization. The main idea is that instead of performing an immediate update to all the variables, we only update variables associate with nonzero elements in x_i. In the following, we derive the lazy update technique for ℓ_1 regularization. As an illustration, Figure 2 shows a simple case where index j has two consecutive zero elements in data vectors chosen at time $t = 1$ and $t = 2$, or

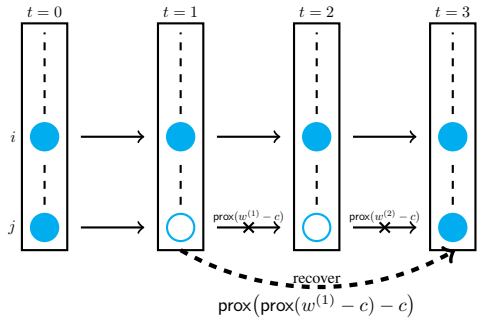

Figure 2. The illustration of lazy-update technique. We count the proximal operations that have been delayed (in this figure there are two) and recover it at once.

$x_{i_1}[j] = x_{i_2}[j] = 0$. So the updates of $w^{(2)}[j]$ and $w^{(3)}[j]$ are:

$$w^{(2)}[j] = \mathsf{prox}\big(w^{(1)}[j] - c\big)$$
$$w^{(3)}[j] = \mathsf{prox}\Big(\mathsf{prox}\big(w^{(1)}[j] - c\big) - c\Big),$$

where $c = \frac{1}{n}\sum_{i=1}^n f_i'(x_i^\intercal \phi_i^{(1)}) x_{i_1}[j]$. Now it remains to calculate the nested proximal operations, which can be effectively calculated by following theorem:

Theorem 6 *Let $g(x) = \eta|x|$, for all $x, c \in \mathbb{R}$ we have:* $\underbrace{\mathsf{prox}_g\Big(\mathsf{prox}_g\big(\cdots \mathsf{prox}_g}_{n \text{ operations}}(x - c)\cdots - c\big) - c\Big) = P_g(x, \eta, c, n)$. *Where P_g is a simple, piecewise linear function.*

Due to the space limit, we left the detailed formulation of P_g in appendix. Upon finishing this paper, we found lazy update for ℓ_1 regularization has also been discussed recently in (Konečnỳ et al., 2016). However, we still include our formal proof here for the completeness of this paper.

3.5. Extension: parallel computing scenario

The fact that doing one step full gradient update is faster than n-step stochastic gradient update dues not only to cache/IO read; similar idea can also be applied to a variety of parallel optimization algorithms, in a more implicit way: when doing full gradient descent, it is trivial to make use of multiprocessing to speedup our program. In contrast, for mini-batch stochastic gradient upgrade when batch size is much smaller than the available CPU cores, many of the computing resources are idle. Although many first order methods have their asynchronous versions that alleviate this problem to some degree (Recht et al., 2011; Leblond et al., 2016; Reddi et al., 2015; Hsieh et al., 2015), because of the inconsistent paces between workers, these algorithms are suboptimal. So if we come back to synchronous updates and given that only full gradient calculation can be significantly accelerated, then the same quantity $T_{\text{seq}}/T_{\text{rand}}$ becomes a deterministic factor that affects how frequently one should perform full batch update.

4. Experimental Results

We compare SAGA++ with SAGA (Defazio, 2014), SVRG (Johnson & Zhang, 2013) and LIBLINEAR (Fan et al., 2008) (proximal Newton method) on solving the ℓ_1-regularized logistic regression:

$$w^* = \arg\min_w \frac{1}{n}\sum_{i=1}^n \log\left(1 + \exp(y_i x_i^\intercal w)\right) + \lambda\|w\|_1, \tag{13}$$

where (x_i, y_i) are feature-response pairs. To make a fair comparison, all the algorithms are implemented based on the LIBLINEAR code base, and we tried to optimize each algorithm. For each outer iteration in SVRG/SAGA++ we choose $m = 1.5n$ inner iterations, this amounts to $\mathbb{E}|\mathcal{B}| = 1.67$ and close to the experiments in (Johnson & Zhang, 2013) where $m = 2n$. The lazy update for ℓ_1 regularization is also implemented for all the variance reduced methods. All the datasets can be downloaded from LIBSVM website.

First, we compare all the algorithms on kddb dataset with different regularization parameters. The results in Figure 3 shows that SAGA++ outperforms other algorithms for all the three choices of parameters. Indeed, $\lambda = 10^{-6}$ (the middle figure) is the best parameter in terms of prediction accuracy, so our comparison covers both large and small λs.

Next, we compare the running time of all the algorithms on three datasets in Figure 4. The results show that SAGA++ is faster than all the competitors on these three datasets. We conclude our experimental results by the following observations: (1) Although SAGA has faster convergence in terms of "number of data access" (see Figure 1-c), SVRG often outperforms SAGA due to faster sequential access. Our algorithm, SAGA++, has a sequential access stage like SVRG, while uses the most up-to-date gradient information at the random update stage, thus outperforms both SVRG and SAGA in all cases. (2) The lazy update (briefly discussed in Section 3.4) accelerates SAGA/SVRG/SAGA++ a lot; without such technique all the variance reduction methods will be much slower than LIBLINEAR. However, with such technique they can outperform the LIBLINEAR implentation of proximal Newton methods.

5. Conclusions and Discussions

We study the unified framework for variance reduction methods with stochastic batch size and prove the linear convergence rate in strongly convex finite sum functions with continuous regularizer. We show that choosing batch size always equals to 1 (equivalent to SAGA) leads to the

Table 1. Dataset statistics. Time for sequential accessing n samples is measured by one computation of full gradient $f'(w)$, while time for random accesses is measured by computing n random gradient components $f_i'(w)$.

Dataset†	Size(GB)	#Sample	#Feature	nnz ratio	Time to access whole data(sec)	
					Sequential	Random
kddb	5.13	19,264,097	29,890,095	9.84e-7	3.91	11.43
avazu	5.04	25,832,830	999,962	1.50e-5	4.14	9.08
criteo	26.74	45,840,617	999,999	3.90e-5	14.07	30.51

†Download from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

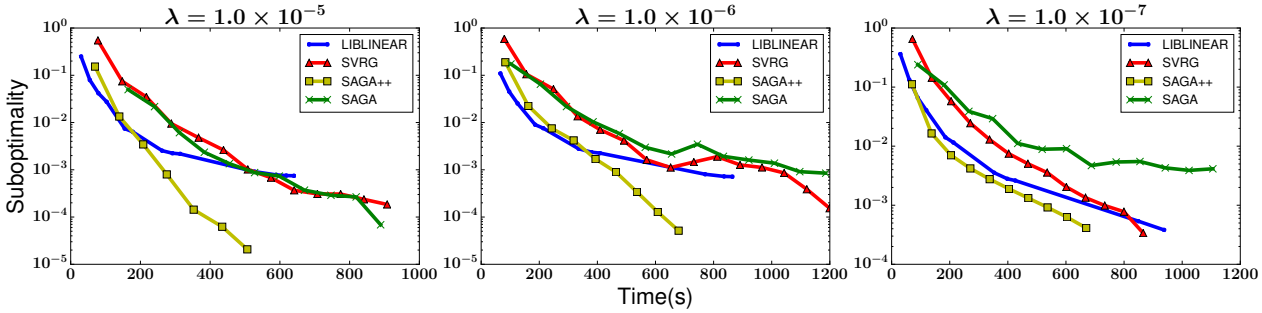

Figure 3. Running time comparison on kddb dataset with different regularization parameters (λ). Result shows that our SAGA++ algorithm is faster than competitors under different regularization parameters.

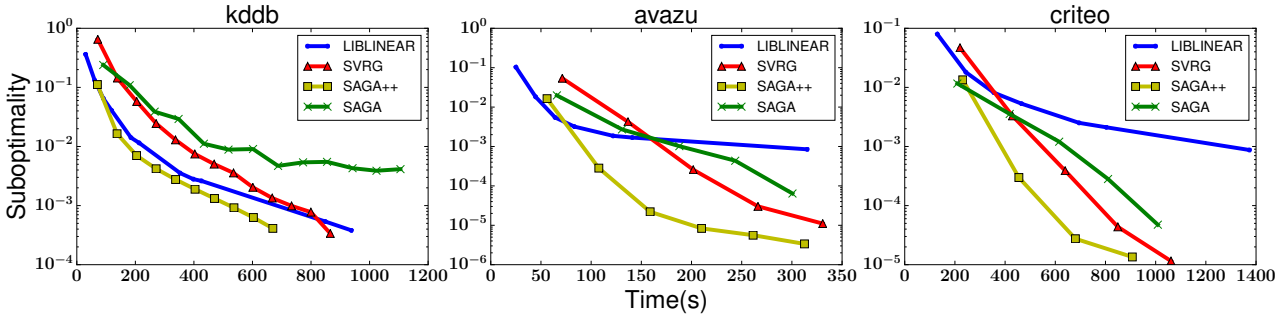

Figure 4. Running time comparison among different data ($\lambda = 1.0 \times 10^{-7}$ for all data). Meta-information can be found in Table 1.

best rate in terms of number of data accesses; however, it is not optimal in terms of running time, so we develop a new SAGA++ algorithm. We demonstrate that SAGA++ outperforms SAGA and SVRG in terms of running time, both in theory and in practice.

One reason that SAGA++ outperforms other VR methods is due to the cache/IO effect, although we only shows in-memory optimization, one can imagine that when data is too large to fit in memory and an out-of-core solver is needed, the overhead of IO will be even more significant. In this setting we would expect an even more greater advantage over SVRG/SAGA. Another important reason is that SAGA++ update its control variate more frequently, making the stochastic gradient a lower variance estimator, so intuitively its more close to gradient descent.

Acknowledgements

The authors acknowledge the support of NSF via IIS-1719097 and the computing resources provided by Google cloud and Nvidia.

References

Allen-Zhu, Z. Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1200–1205. ACM, 2017.

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer, 2012.

Csiba, D. and Richtárik, P. Primal method for erm with flexible mini-batching schemes and non-convex losses. *arXiv preprint arXiv:1506.02227*, 2015.

De, S., Yadav, A., Jacobs, D., and Goldstein, T. Big batch sgd: Automated inference using adaptive batch sizes. *arXiv preprint arXiv:1610.05792*, 2016.

Defazio, A. *New Optimization Methods for Machine Learning*. PhD thesis, PhD thesis, Australian National University, 2014.

Defazio, A., Bach, F., and Lacoste-Julien, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pp. 1646–1654, 2014.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Harikandeh, R., Ahmed, M. O., Virani, A., Schmidt, M., Konečnỳ, J., and Sallinen, S. Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems*, pp. 2251–2259, 2015.

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

Hsieh, C.-J., Yu, H.-F., and Dhillon, I. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *International Conference on Machine Learning*, pp. 2370–2379, 2015.

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

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Konecnỳ, J. and Richtárik, P. Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666*, 2(2.1): 3, 2013.

Konečnỳ, J., Liu, J., Richtárik, P., and Takáč, M. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Asaga: asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*, 2016.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670. ACM, 2014.

Lin, H., Mairal, J., and Harchaoui, Z. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pp. 3384–3392, 2015.

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

Qu, Z., Richtárik, P., and Zhang, T. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Advances in neural information processing systems*, pp. 865–873, 2015.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Reddi, S. J., Hefny, A., Sra, S., Poczos, B., and Smola, A. J. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pp. 2647–2655, 2015.

Richtárik, P. and Takáč, M. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, pp. 1–30.

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

Takác, M., Bijral, A. S., Richtárik, P., and Srebro, N. Mini-batch primal and dual methods for svms. In *ICML (3)*, pp. 1022–1030, 2013.