Tracking the gradients using the Hessian: A new look at variance reducing stochastic methods

Robert M. Gower
robert.gower@telecom-paristech.fr
LTCI, Télécom Paristech,
Université Paris-Saclay[1]

Nicolas Le Roux
nicolas.le.roux@gmail.com
Google Brain

Francis Bach
francis.bach@inria.fr
INRIA - ENS - PSL Research University

Abstract

Our goal is to improve variance reducing stochastic methods through better control variates. We first propose a modification of SVRG which uses the Hessian to track gradients over time, rather than to recondition, increasing the correlation of the control variates and leading to faster theoretical convergence close to the optimum. We then propose accurate and computationally efficient approximations to the Hessian, both using a diagonal and a low-rank matrix. Finally, we demonstrate the effectiveness of our method on a wide range of problems.

1 Introduction

Many machine learning problems can be written as the minimization of a loss over a set of samples. Though this set can be infinite, it is in most cases finite, in which case the problem becomes the minimization of the average loss over N samples, i.e.

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} f_i(\theta) = \arg\min_{\theta \in \mathbb{R}^d} F(\theta), \qquad (1)$$

where $f_i(\theta)$ is the loss incurred by parameters θ for the i-th sample. In this work, we make the assumption that each function f_i is strongly convex.

A common way of solving Eq. (1) is through a stochastic gradient method (Robbins and Monro, 1951). Starting from θ_0, the standard stochastic method samples i uniformly at random in $\{1, \ldots, N\}$ then computes $\theta_{t+1} = \theta_t - \gamma_t \frac{\partial f_i(\theta)}{\partial \theta}|_{\theta=\theta_t}$ where γ_t can be a scalar or a full matrix. To unclutter notation, we will use g to denote the gradient, i.e. $g_i(\theta_t) = \frac{\partial f_i(\theta)}{\partial \theta}|_{\theta=\theta_t}$.

Stochastic gradient updates are not convergent for constant stepsizes γ_t since the variance of $g_i(\theta_t)$ does not converge to 0 (Schmidt, 2014). Achieving convergence requires a decreasing schedule which impacts the convergence speed. By reducing the variance of these updates, one can hope to achieve convergence with larger or even constant stepsizes and thus obtain a faster rate. We now review the most popular stochastic variance reducing methods.

SAG (Le Roux et al., 2012; Schmidt et al., 2013) was the first stochastic method to achieve a linear convergence rate for strongly convex problems over a finite dataset. It required a storage linear in N and used biased parameter updates. SDCA (Zhang et al., 2013; Shalev-Shwartz and Zhang, 2013) is a similar method solving a dual minimization problem for an important subclass of such problems. Compared to SAG which is adaptive, SDCA requires the knowledge of the strong convexity constant.

SVRG (Johnson and Zhang, 2013) achieves a similar rate with a storage requirement independent of N, albeit at the expense of a constant factor increase in the computational cost. SVRG also requires an extra parameter besides the stepsize: the frequency at which the true gradient is recomputed. This is also the first work to provide the variance reduction interpretation. In this paper, we will mostly consider extensions of SVRG.

SAGA (Defazio et al., 2014) modifies SAG to get unbiased updates, leading to better convergence rates. Using a proximal algorithm, SAGA also works for non-smooth regularizers.

MISO (Mairal, 2013) is a majorization-minimization algorithm achieving a linear convergence rate on strongly convex problems but it can also be used to

[1]Work done while at Inria, SIERRA team.

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84.

solve nonconvex problems under some conditions.

All of these methods reduce variance by incorporating knowledge about gradients on all datapoints at every timestep rather than only using the gradient for the sampled datapoint. Where they differ is in which gradients they use and how they are incorporated.

This paper builds up on all these methods and makes the following contributions:

- We recall in Section 2 that most of these methods can be seen as using control variates, an observation made by Defazio et al. (2014), then highlight that none of the control variates currently used track the gradients.
- We propose in Section 3 a new control variate that tracks the gradient by building a linear model which is based on the Hessian matrix. This second-order information is used to further reduce the variance of the parameter updates, and not as a preconditioner.
- Since calculating Hessians typically scales as $O(d^2)$, we propose several new approximations in Section 4 based on diagonal or low-rank matrices. Such techniques are usually only dedicated to preconditioning.
- We present in Section 5 a faster theoretical convergence rate than that of SVRG within a neighborhood of the solution.
- Finally, in our experiments in Section 6, we show a significant improvement over SVRG.

2 Control variates

We now recall that most variance reduced methods use control variates, a well-known technique in Monte-Carlo simulation (see, e.g., Rubinstein and Kroese (2016)) designed to reduce the variance of the estimate of the expectation of a random variable. Let us start by describing this technique.

We try to estimate the unknown expectation $\bar{x}$ of a random variable x and that we have access to another random variable, z, whose expectation $\bar{z}$ is known. Then the quantity $x_z = x - z + \bar{z}$ has expectation $\bar{x}$ and variance $V[x_z] = V[x] + V[z] - 2\text{Cov}(x, z)$ where $V[]$ is the variance and Cov[] the covariance. $V[x_z]$ is lower than $V[x]$ whenever z is sufficiently positively correlated with x and the variance reduction is larger when the control variate is more correlated with the random variable.

Using the true gradient $g(\theta_t) = \frac{1}{N}\sum_i g_i(\theta_t)$, batch methods achieve a linear convergence rate, albeit with an update computational cost linear in N. Stochastic methods estimate the true gradient through a single[1] sample $g_i(\theta_t)$, leading to a slower convergence rate but cheaper updates. The goal is thus to estimate the true gradient as accurately as possible while still maintaining update costs independent of N. As the true gradient is the expectation of the individual gradients, control variates can be applied.

More specifically, using a control variate to reduce the variance of the gradient estimation means replacing the quantity $g_i(\theta_t)$ with

$$\tilde{g}_i(\theta_t) = g_i(\theta_t) - z_i(\theta_t) + \frac{1}{N}\sum_j z_j(\theta_t) \;, \qquad (2)$$

transforming the stochastic gradient updates into

$$\theta_{t+1} = \theta_t - \gamma_t\Big(g_i(\theta_t) - z_i(\theta_t) + \frac{1}{N}\sum_j z_j(\theta_t)\Big) \;. \qquad (3)$$

The new gradient estimate $\tilde{g}_i(\theta_t)$ is an unbiased estimator of the true gradient $g(\theta_t)$ with lower variance than $g_i(\theta_t)$ when $z_i(\theta_t)$ is positively correlated with $g_i(\theta_t)$.

The main difference between existing variance reducing gradient methods lies in the chosen $z_i(\theta_t)$. SAGA, for instance, chooses $z_i(\theta_t) = g_i(\bar{\theta}_i)$ the last gradient computed for sample i. Using such a control variate requires a storage linear in N. SVRG uses $z_i(\theta_t) = g_i(\bar{\theta}_t)$, the gradient computed at a fixed $\bar{\theta}_t$ for all samples. This allows for the computation of the control variate while only having to store $\bar{\theta}_t$. As it is the same for all datapoints, the storage requirement is independent of N.

One commonality to all existing methods is that each control variate remains constant until a new gradient has been computed for that datapoint. In particular, this means that control variates can easily become outdated if the gradients change quickly, leading to lower correlation and thus higher variance of the estimator. Since the variance of the estimator is directly linked to the convergence speed, improving correlation will lead to faster convergence. In the next section, we will show how to modify SVRG to make use of second-order approximations, maintaining highly correlated control variates longer than in SVRG.

3 Tracking gradients with second-order control variates

We now show how we can modify all stored gradients with a cost independent of N by using second-order information about each individual function.

[1] Or a small number in the case of minibatch gradient methods.

All algorithms of the previous section used for $z_i(\theta_t)$ the gradient computed for the same sample at a previous parameter $\bar{\theta}$. In particular, $z_i(\theta_t)$ does not change when the ith datapoint is not sampled. While this is efficient close to convergence when θ does not move much, correlation might be low early on, leading to slower convergence.

In particular, the convergence rate of SAGA is proportional to $\left(1 - \frac{\mu}{2(\mu N + L_{\max})}\right)$, with μ the strong-convexity constant and $L_{\max}$ the maximum smoothness constant of the individual f_i's.[2] It is crucial to note that $L_{\max}$ is not the same as L, the smoothness constant of the full function F. More precisely, we have $L_{\max}/d \leq L \leq L_{\max}$. When all individual gradients are equal we have that $L = L_{\max}$ and SAGA is exactly N times faster than full gradient descent. On the other hand, when the individual gradients are uniformly distributed over the whole space, we have $L = L_{\max}/d$ and the speed improvement of SAGA depends on N/d.

When $\mu/L_{\max}$ is much smaller than $1/N$, the poor condition number means that θ does not move much and SAGA should be close to batch gradient descent[3]. Indeed, the rate is close to $(1 - \mu/L_{\max})$ which is similar to $(1 - \mu/L)$, the convergence rate of batch gradient descent. However, the difference between L and $L_{\max}$ means that, even in that regime, the rate of batch gradient descent can be much better. This is because batch gradient descent can make use of a stepsize of $1/L$, potentially much larger than the stepsize of $\frac{1}{2(\mu N + L_{\max})}$ of SAGA.

When $\mu/L_{\max}$ is larger than $1/N$, the convergence rate of SAGA is close to $1 - 1/2N$, which is much smaller than the rate of convergence of batch gradient descent. This is because, in the well-conditioned case, θ changes quickly, leading to poorly correlated control variates and slower convergence relative to batch gradient descent.

Thus, finding control variates that have a greater correlation with the gradient would allow these algorithms to maintain a fast convergence for larger values of N and potentially use larger stepsizes. Additionally, SVRG requires regular recomputation of $\bar{\theta}_t$ to maintain its convergence rate. Maintaining a high correlation for longer could decrease this required frequency of recomputation.

Algorithm 1 describes the original SVRG method (Johnson and Zhang, 2013).

[2]For linear regression, we have $L_{\max} = \max_i \|x_i\|^2 = R^2$ the radius of the input data.

[3]The resulting algorithm would be N times faster since an update only requires looking at a single datapoint compared to the full dataset for batch gradient descent.

Algorithm 1 SVRG

Parameter: Functions f_i for $i = 1, \ldots, N$
Choose $\bar{\theta}_0 \in \mathbb{R}^d$ and stepsize $\gamma > 0$
for $k = 0, \ldots, K-1$ **do**
  Calculate $g(\bar{\theta}_k) = \frac{1}{N}\sum_{j=1}^N g_j(\bar{\theta}_k)$, $\theta_0 = \bar{\theta}_k$
  for $t = 0, 1, 2, \ldots, T-1$ **do**
    $i \sim \mathcal{U}[1, N]$
    $\theta_{t+1} = \theta_t - \gamma\Big(g_i(\theta_t) - g_i(\bar{\theta}_k) + g(\bar{\theta}_k)\Big)$
  $\bar{\theta}_{k+1} = \theta_T$
Output $\bar{\theta}_K$

Algorithm 2 SVRG2 (SVRG with tracking)

Parameter: Functions f_i for $i = 1, \ldots, N$
Choose $\bar{\theta}_0 \in \mathbb{R}^d$ and stepsize $\gamma > 0$
for $k = 0, \ldots, K-1$ **do**
  Calculate $g(\bar{\theta}_k) = \frac{1}{N}\sum_{j=1}^N g_j(\bar{\theta}_k)$, $\theta_0 = \bar{\theta}_k$
  Calculate $\underline{H(\bar{\theta}_k) = \frac{1}{N}\sum_{j=1}^N H_j(\bar{\theta}_k)}$
  for $t = 0, 1, 2, \ldots, T-1$ **do**
    $i \sim \mathcal{U}[1, N]$
    $\theta_{t+1} = \theta_t - \gamma\Big(g_i(\theta_t) - g_i(\bar{\theta}_k) + g(\bar{\theta}_k)\Big)$
      $\underline{-\gamma\big(H(\bar{\theta}_k) - H_i(\bar{\theta}_k)\big)(\theta_t - \bar{\theta}_k)}$
  $\bar{\theta}_{k+1} = \theta_T$
Output $\bar{\theta}_K$

We now modify Algorithm 1 by using control variates based on the first order Taylor expansion of the stochastic gradient. Rather than $z_j(\theta_t) = g_j(\bar{\theta})$, we will use:

$$z_j(\theta_t) = g_j(\bar{\theta}) + H_j(\bar{\theta})(\theta_t - \bar{\theta}), \tag{4}$$

where $H_j(\bar{\theta})$ is the Hessian of f_j taken at $\bar{\theta}$. The new control variate $z_j(\theta_t)$ now depends on θ_t, which is not the case for the control variate used in SVRG.

Plugging Eq. (4) into Eq. (3) yields the following update:

$$\theta_{t+1} = \theta_t - \gamma_t\Big(g_i(\theta_t) - g_i(\bar{\theta}) + \frac{1}{N}\sum_j g_j(\bar{\theta})\Big) - \gamma_t\Big(\frac{1}{N}\sum_j H_j(\bar{\theta}) - H_i(\bar{\theta})\Big)(\theta_t - \bar{\theta})\,. \tag{5}$$

The resulting method, SVRG2, is presented in Algorithm 2. The only changes compared to Algorithm 1 are the computation of $H(\bar{\theta})$ and the terms involving the Hessian in the update (underlined). The theoretical analysis of Algorithm 2 for generalized linear models is carried out in Section 5.

It is apparent that, if the f_i's are exactly quadratic, then $g_i(\bar{\theta}) + H_i(\bar{\theta})(\theta_t - \bar{\theta}) = g_i(\theta_t)$ and the update is the same as batch gradient descent, albeit with a computational cost independent of N. The goal is thus to

yield an update as close as possible to that of batch gradient descent without paying the price of recomputing all gradients every time.

Algorithm 2 requires the computation of the full Hessian every time a new $\bar{\theta}$ is chosen and does two matrix-vector products for each parameter update. The cost of computing the full Hessian is $O(Nd^2)$ and the cost of each update is $O(d^2)$. Since T, the number of updates before recomputing the full gradient, is typically a multiple of N, the average cost per update is $O(d^2)$, which is d times bigger than all first-order stochastic methods. As this cost is prohibitive for everything but low-dimensional problems, we need to resort to using approximations of the Hessian.

4 Approximations of the Hessian

Algorithm 2 uses the following quantities:

- $H_i(\bar{\theta})$ needed for multiplying with the current θ_t;
- $\sum_j H_j(\bar{\theta})$ needed for multiplying with the current θ_t;
- $H_i(\bar{\theta})\theta_i$, $\sum_j g_j(\bar{\theta})$ and $\sum_j H_j(\bar{\theta})\bar{\theta}$ which are all vectors.

The first and second terms cannot be stored for large dimensional problems and, even in that case, the computational cost will be prohibitive. We thus need to resort to approximations $\hat{H}_j(\bar{\theta})$ of each Hessian $H_j(\bar{\theta})$ such that their sum $\hat{H}(\bar{\theta})$ is also easy to store.

This restriction makes us focus on linear projections of the Hessians, and two in particular: diagonal approximations and projections onto a low-rank subspace. Note that both of these classes of approximations include the special case $H_i = 0$, which is the original SVRG.

4.1 Diagonal approximation

We propose here several diagonal approximations of each Hessian leading to a storage cost of $O(d)$.

Diagonal of the Hessian: By minimizing the Frobenius distance with the true Hessian, the diagonal of the Hessian tries to approximate the true Hessian in all directions of the space. This is in general very conservative as we care mostly about approximating the Hessian in the directions $\theta_t - \bar{\theta}$.

Using the secant equation: Since the approximation is used in the term $\hat{H}_i(\bar{\theta})(\theta_t - \bar{\theta})$, we want $\hat{H}_i(\bar{\theta})(\theta_t - \bar{\theta})$ to be a good approximation of $H_i(\bar{\theta})(\theta_t - \bar{\theta})$. One can use the secant equation and set

$$\hat{H}_i(\bar{\theta}) = \frac{H_i(\bar{\theta})(\theta_t - \bar{\theta})}{\theta_t - \bar{\theta}} \approx \frac{g_i(\theta_t) - g_i(\bar{\theta})}{\theta_t - \bar{\theta}} \ ,$$

where the division between two vectors is to be taken elementwise. However, this leads to unstable updates as no guarantee is given outside of one direction.

Robust secant equation: We can robustify the secant equation by minimizing the average squared-ℓ_2 distance within a small ball around the previous direction. In other words, we seek

$$\hat{H} = \operatorname*{arg\,min}_{Z \in \mathbb{R}^{d\times d}} \int_{\xi} \left\| (Z - H_i)(\theta_t - \bar{\theta} + \xi) \right\|^2 p(\xi) d\xi.$$

Assuming $\xi \sim \mathcal{N}(0, \sigma^2 I)$, we get

$$\begin{aligned} \hat{H} &= \frac{(\theta_t - \bar{\theta}) \odot H_i(\bar{\theta})(\theta_t - \bar{\theta}) + \sigma^2 \mathrm{diag}(H_i(\bar{\theta}))}{(\theta_t - \bar{\theta}) \odot (\theta_t - \bar{\theta}) + \sigma^2} \\ &\approx \frac{(\theta_t - \bar{\theta}) \odot (g_i(\theta_t) - g_i(\bar{\theta})) + \sigma^2 \mathrm{diag}(H_i(\bar{\theta}))}{(\theta_t - \bar{\theta}) \odot (\theta_t - \bar{\theta}) + \sigma^2}. \quad (6) \end{aligned}$$

This is an interpolation between the two previous approximations. In particular, $\sigma^2 = +\infty$ recovers the diagonal of the Hessian while $\sigma^2 = 0$ recovers the secant equation. For a fixed σ^2, larger parameters updates at the beginning of optimization will make this approximation close to the secant equation while, close to convergence, this approximation will converge towards the diagonal of the Hessian due to smaller parameter updates.

4.2 Low-rank approximation: Curvature Matching

Another possibility is to build a low rank approximation of the Hessian by using a low dimensional embedding. For brevity let $H_i := H_i(\bar{\theta})$ and $H := H(\bar{\theta})$.

Consider a matrix $S \in \mathbb{R}^{d\times k}$ where $k \ll d, N$. The matrix S may be random. Calculating the *embedded* Hessian $S^\top H_i S$ gives us a low rank approximation of H_i using the following model:

$$\begin{aligned} \hat{H}_i = \operatorname*{arg\,min}_{X \in \mathbb{R}^{d\times d}} & \|X\|^2_{F(H)} \\ \text{subject to} \quad & S^\top X S = S^\top H_i S, \quad (7) \end{aligned}$$

where $\|\hat{H}\|^2_{F(H)} := \mathrm{Tr}\left(\hat{H}^\top H \hat{H} H\right)$ is the weighted Frobenius norm. The solution to the above is a rank k matrix given by

$$\hat{H}_i = HS(S^\top HS)^\dagger S^\top H_i S (S^\top HS)^\dagger S^\top H \in \mathbb{R}^{d\times d}, \quad (8)$$

where $M^\dagger$ is the pseudoinverse of M. We refer to methods based on the Hessian approximation presented in Eq. (8) as the *Curvature Matching* methods since the approximation has the same curvature as the true Hessian over the subspace spanned by S.

Our use of a Hessian weighted Frobenius norm in (7)

is largely inspired on quasi-Newton methods. Indeed, during the development of quasi-Newton methods it was observed that using the Hessian as a weighting matrix in the Frobenius norm gave rise to a more efficient method (Goldfarb, 1970), namely the BFGS method. Our findings mirror that of the quasi-Newton literature in that our empirical tests showed that using the Hessian weighted Frobenius norm is more efficient than using the standard Frobenius norm.

Note that, to compute Eq. (8), we only need to store the $d \times k$ matrix HS and calculate $S^\top H_i S$. The advantages of this approach are numerous and include: (a) Reduced memory storage, (b) Hessian-vector products $\hat{H}_i v$ can be computed efficiently at a cost of $O(k(2d+3k))$ which is linear in d, (c) the approximation $\hat{H}_i$ is the result of applying a linear operator on H_i and consequently the expectation of $\hat{H}_i$ can be computed efficiently.

This last item is of particular importance for maintaining an unbiased estimator. Indeed, taking the expectation over (8), we have that

$$\begin{aligned}\mathbf{E}_i\left[\hat{H}_i\right] &\overset{(8)}{=} HS(S^\top HS)^\dagger S^\top \mathbf{E}_i\left[H_i\right] S(S^\top HS)^\dagger S^\top H \\ &= HS(S^\top HS)^\dagger S^\top HS(S^\top HS)^\dagger S^\top H \\ &= HS(S^\top HS)^\dagger S^\top H, \end{aligned} \tag{9}$$

where in the last step we used that $M^\dagger M M^\dagger = M^\dagger$ for all matrices M. From Eq. (9) we see that the expectation of $\hat{H}_i$ is equal to the resulting approximation were we to use H in place of H_i in Eq. (8). Furthermore (9) is a low rank matrix which can be stored and computed efficiently since we only need to calculate the action of the true Hessian HS.[4]

Specifically, in the outer iterations, we can compute and store the $d \times k$ Hessian *action* matrix $A = \frac{1}{N}\sum_{j=1}^N H_j(\bar{\theta})S$ and the $k \times k$ Hessian *curvature* matrix $C = (S^\top \frac{1}{N}\sum_{j=1}^N H_j(\bar{\theta})S)^{\dagger/2}$ then use A and C to perform the necessary matrix vector products. See Algorithm 3 for our implementation. Again we underline the computations that need to be added to the SVRG algorithm to produce our new algorithm.

While S can be any matrix, including a random one drawn from some distribution, we found the method performed best when the construction of S was based on the step directions d_t for $t = 0, \ldots, T-1$ taken from the inner loop. The reasoning behind this choice is that, since ultimately we only require the action of the Hessian approximation along the directions $\theta_t - \bar{\theta}$, the matrix S should be chosen in such a way that our Hessian approximation is accurate along these directions. Thus we will construct the embedding matrix

[4] With automatic differentiation, it costs only k times the cost of evaluating the function $F(\theta)$ to compute HS.

Algorithm 3 CM: Curvature Matching

Parameter: Functions f_i for $i = 1, \ldots, N$
Choose $\bar{\theta}_0 \in \mathbb{R}^d$ and stepsize $\gamma > 0$
for $k = 0, \ldots, K-1$ **do**
  Calculate $g(\bar{\theta}_k) = \frac{1}{N}\sum_{j=1}^N g_j(\bar{\theta}_k)$, $\theta_0 = \bar{\theta}_k$
  Calculate $A = \frac{1}{N}\sum_{j=1}^N H_j(\bar{\theta})S$, $C = (S^\top A)^{\dagger/2}$.
  Generate $S \in \mathbb{R}^{d\times k}$, calculate $\bar{S} = SC$.
  Normalize the Hessian action $\bar{A} = AC$.
  for $t = 0, 1, 2, \ldots, T-1$ **do**
    $i \sim \mathcal{U}[1, N]$
    $d_t = g_i(\theta_t) - g_i(\bar{\theta}_k) + g(\bar{\theta}_k)$
      $-\bar{A}\bar{S}^\top H_i(\bar{\theta}_k)\bar{S}\bar{A}^\top(\theta_t - \bar{\theta}_k) + \bar{A}\bar{A}^\top(\theta_t - \bar{\theta}_k)$
    $\theta_{t+1} = \theta_t - \gamma d_t$
  $\bar{\theta}_{k+1} = \theta_T$
  Output $\bar{\theta}_K$

S by again taking inspiration from the BFGS (Goldfarb, 1970) and L-BFGS (Nocedal, 1980) methods and column concatenate previous step directions. Specifically, since there are many step directions and sequential step directions are often correlated, we arranged the T step directions into k sets. We used the average over each of these k sets as a column of S. That is

$$S = [\bar{d}_0, \ldots, \bar{d}_{k-1}] \quad \text{where} \quad \bar{d}_i = \frac{k}{T}\sum_{j=\frac{T}{k}i}^{\frac{T}{k}(i+1)-1} d_j, \tag{10}$$

where we assume, for simplicity, that k divides T.

Computational complexity: Algorithm 3 can be applied to large-scale problems due to its reduced complexity. Indeed, using automatic differentiation (Walther, 2008; Christianson, 1992) the cost of computing each one of the matrix-matrix products $H_j(\bar{\theta})S$ is $O(k \cdot \text{eval}(f_j))$, where $\text{eval}(f_j)$ is the cost of evaluating the function f_j. Furthermore, evaluating a single gradient g_j also costs $O(\text{eval}(f_j))$. The matrix product $S^\top A$ costs $O(k^2 d)$ to compute, while the products AC and SC cost dk^2. Computing the square and pseudoinverse in $C = (S^\top A)^{\dagger/2}$ costs $O(k^3)$, leading to a total for the outer iteration of

$$O(k\cdot\sum_{j=1}^n \text{eval}(f_j) + k^2 d + k^3) = O(k\cdot\text{eval}(f) + k^2 d + k^3).$$

The main cost in the inner iteration is computing $H_i(\bar{\theta}_k)S$ and a handful of matrix-vector products where the matrices have dimensions $k\times d$ or $d\times k$. The total cost of each inner iteration is $O(k\cdot\text{eval}(f_i) + dk^2)$. Overall, the cost of computing an iteration (inner or outer) of Algorithm 3 is linear in d and comes at a k-fold increase of the cost of computing an SVRG iteration. In our experiments, we used $k = 10$.

4.3 Low-rank approximation: Action Matching

In the previous section we developed a low rank update based on curvature matching, though ultimately we required the action of the true Hessian HS to compute the resulting Hessian approximation (8). With the action of the Hessian we can also build an approximation of the Hessian using an *action matching* model:

$$\hat{H}_i = \arg\min_{X\in\mathbb{R}^{d\times d}} \|X\|^2_{F(H)} \quad \text{subject to} \quad XS = H_iS, \quad X = X^\top. \tag{11}$$

In other words, $\hat{H}_i$ is the symmetric matrix with the smallest norm which matches the action of the true Hessian. The solution to the above is a rank $2k$ matrix given by

$$\hat{H}_i = HS(S^THS)^{-1}S^\top H_i\left(I - S(S^THS)^{-1}S^\top H\right) + H_iS(S^THS)^{-1}S^\top H. \tag{12}$$

Differently from the curvature matching model (7), here we have had to explicitly enforce the symmetry of the Hessian approximation as a constraint (11) as the solution of the unconstrained model might not be symmetric.

This approximation is a generalization of the Davidson-Fletcher-Powell (Fletcher and Powell, 1963; Davidon, 1968) update. This update, and a regularized inverse of this update, have already been used in an optimization context (Gower and Gondzio, 2014) but only as a preconditioner.

Again, we have that the approximation (12) is the result of applying a linear function to H_i. Thus, taking the expectation, we have that

$$\begin{aligned}\mathbf{E}_i\left[\hat{H}_i\right] &= HS(S^THS)^{-1}S^\top\mathbf{E}_i\left[H_i\right] \\ &\quad + \mathbf{E}_i\left[H_i\right]S(S^THS)^{-1}S^\top H \\ &\quad - HS(S^THS)^{-1}S^\top\mathbf{E}_i\left[H_i\right]S(S^THS)^{-1}S^\top H \\ &= HS(S^THS)^{-1}S^\top H,\end{aligned}$$

which is the same as (9) and thus can be efficiently computed. The method based on (11) can be implemented in a similar way to Algorithm 3. The total complexity of the algorithm based on (11) is the same as the complexity of Algorithm 3, though the hidden constant in the complexity is a bit larger as there is an additional matrix-vector product.

4.4 Comparing approximations

Both the diagonal and the low-rank approximations have a free parameter: σ^2 for the diagonal approximation and the rank k for the low-rank approximation. These two parameters make different tradeoffs. σ^2 controls the robustness of the approximation: a larger value is more stable but likely to decrease the convergence rate. k controls the complexity of the approximation: a larger value leads to a better convergence rate in terms of updates but with a larger computational cost per update. Though this was not explored in this work, it seems natural to combine these two approximations.

5 Theoretical Analysis and Discussion

Convergence rates: We now provide a convergence analysis for generalized linear models when an approximation of the Hessian is used. The full proofs of the results below may be found in the Appendix. We make the following assumptions:

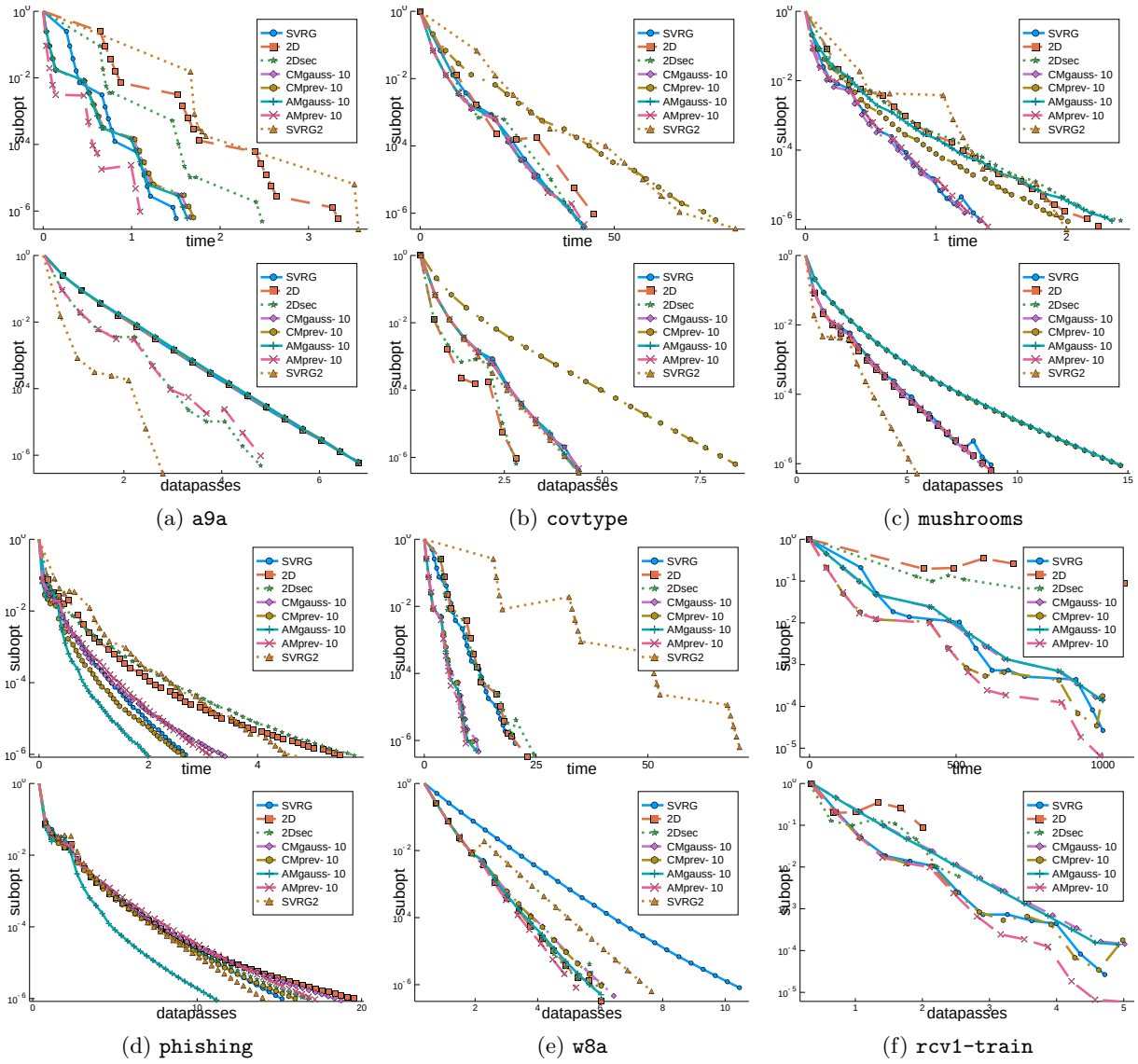
- We consider a compact set Θ included in the ball of center θ^* and radius D, where θ^* is the minimizer of $F(\theta) = \frac{1}{N}\sum_{i=1}^N f_i(\theta)$.
- F is μ-strongly convex and L-smooth on Θ.
- Each f_i is of the form $f_i(\theta) = \varphi_i(x_i^\top\theta)$ with $\|x_i\|^2 \leqslant R^2$, and $\varphi_i''(x_i^\top\theta) \in [\alpha, 1]$, $\varphi_i'''(x_i^\top\theta) \leqslant \beta$ for all $\theta \in \Theta$. In particular, this means that we can use $L_{\max} = R^2$.
- We consider the control variate $z_i(\theta) = f_i'(\bar{\theta}) + H_i(\theta - \bar{\theta})$, where H_i is an approximation of the Hessian $f_i''(\bar{\theta})$, with a relative error so that $\frac{1}{N}\sum_{i=1}^N (f_i''(\bar{\theta}) - H_i)^2 \preccurlyeq R^2\eta\frac{1}{N}\sum_{i=1}^n f_i''(\bar{\theta})$. Note that $\eta = 1$ if we take $H_i = 0$ (plain SVRG) and $\eta = 0$ if we use the exact Hessian.

We obtain the following proposition which shows that we get the actual condition number L/μ rather than $L_{\max}/\mu$ (which is the one for SVRG) if we are close enough to the global optimum and the Hessians are sufficiently well approximated:

Proposition 1 *Assume* $D^2 \leqslant \frac{L\alpha}{8\beta^2R^2}$, $\gamma = 1/(4L)$ *with* $\eta \leqslant \frac{L\alpha}{8\beta^2}$. *Then, after a single epoch of SVRG2 with* $T \geqslant \frac{16L}{\mu}$, *the error is reduced by a factor of* $3/4$.

The result above implies that, in order to reach a precistion ε, we need $K = O(\log\frac{1}{\varepsilon})$ epochs of SVRG2, with an overall number of accesses to gradient and Hessian oracles less than $O(N + \frac{L}{\mu}\log\frac{1}{\varepsilon})$, once we are close enough to the optimum (but note that the bound on D does not depend on μ). In the Appendix, we also show that our algorithm is robust even far away from optimum with a step-size $\gamma \propto 1/R^2$.

Difference to reconditioning: To our knowledge, this is the first use of the Hessian to track gradients over time rather than to recondition. While using the



(a) a9a (b) covtype (c) mushrooms

(d) phishing (e) w8a (f) rcv1-train

Figure 1: Performance of various SVRG-based methods on multiple LIBSVM problems: (a) a9a (b) covtype (c) mushrooms (d) phishing (e) w8a (f) rcv1-train.

Hessian to track the gradients does not remove the dependency on the condition number, it has several advantages. First, since there is no matrix inversion, it is much more robust to approximations of the true Hessian, as shown in the theoretical analysis above. Second, while classical second-order methods only enjoy superlinear convergence close to the optimum, tracking gradients improves convergence in a larger ball around the optimum.

This does not preclude the use of reconditioning and we believe the Hessian could be used for both purposes simultaneously. We do not know if the best Hessian approximation is the same for both uses.

Difference to non-uniform sampling: Non-uniform sampling aims to close the gap between L_{max} and $\frac{1}{n}\sum_i L_i$ (Kern and Gyorgy (2016) and Schmidt et al. (2015)). Our method closes the gap between L_{max} and L. These two quantities, $\frac{1}{n}\sum_i L_i$ and L, can be very different. For instance, for a linear least squares objective $f(w) = 1/(2n)||Xw - b||_2^2$ we have that $\frac{1}{n}\sum_i L_i = (1/n)\mathrm{Tr}\left(XX^\top\right)$ while $L = \frac{1}{n}\lambda_{\max}(XX^\top)$, thus L can be upto n times smaller than $\frac{1}{n}\sum_i L_i$.

6 Experiments

Algorithms. We empirically tested the impact of using second-order information for the different Hessian approximations. In particular, we tried SVRG, the original SVRG algorithm, SVRG2, which tracks the gradi-

ents using the full Hessian, `2D`, which tracks the gradients using the diagonal of the Hessian, `2Dsec`, which tracks the gradients using the robust secant equation (6), `CM`, which tracks the gradients using the low-rank curvature matching approximation of the Hessian (8), and `AM` which uses the low-rank action matching approximation of the Hessian (12) to track the gradients. We also tested two variants of each of the low rank approximations, namely `AMgauss, CMgauss` and `AMprev, CMprev`. For the `AMgauss` and `CMgauss` methods, the matrix $S \in \mathbb{R}^{n\times k}$ in (8) and (12) is a randomly generated Gaussian matrix. For the `AMprev` and `CMprev` methods, we construct S using previous search directions according to (10).

All the data we use are binary classification problems taken from LIBSVM, and we use a logistic loss with a squared-ℓ_2 regularizer. We set the regularization parameter to $\lambda = \max_{i=1,\dots,N}\|x_i\|_2^2/4N$ in all experiments. In each figure, the y-axis is the relative suboptimality gap given by $[f(\theta) - f(\theta^*)]/[f(\theta_0) - f(\theta^*)]$ on a log scale.

Convergence rate was observed both in terms of passes through the data and in terms of wall clock time, to account for the higher computational cost of our proposed methods. Due to the high computational cost of `SVRG2`, we did not perform experiments on `SVRG2` for problems with dimension (number of features) over 5000.

Choice of the stepsize. Since convergence is guaranteed for batch gradient descent with a larger stepsize than for variance reducing stochastic methods, our hope is that using second-order information will allow the use of larger stepsizes while maintaining convergence. Thus, we ran a grid search over the stepsize for each method, trying values of $2^a/L_{\max}$ for $a = 10, 7, \dots, -7, -9$, where $L_{\max} := \max_{i=1,\dots,N}\|x_i\|_2^2 + \lambda$.

We expected our proposed covariates to provide better approximations to the true gradient and thus allow for greater stepsizes. This is the case in general, albeit not by a large amount, as can be seen in Table 1. Our proposed methods often allowed larger stepsizes, with the `SVRG2` method typically allowing the largest stepsize.

Results. Figure 1 shows the results on multiple datasets. We see that, in terms of passes through the data, all tracking methods consistently outperform `SVRG`, often by a large margin. In particular the `SVRG2` method often has the best performance in terms of datapasses with the exception of the `covtype` problem where the two methods based on diagonal approximations `2D` and `2Dsec` have the best performance[5]. In terms of time taken, the `AMgauss` and `AMprev` are the most efficient methods, which indicates the superiority of the Hessian approximations based on action matching (11). For all these experiments, we chose a rank $k = 10$.

[5]The hyperparamter σ^2 was set to $\sigma^2 = 0.01$.

	a9a	covtype	gisette	madelon
SVRG	2^8	2^9	2^8	2^7
2D	2^8	2^{10}	2^8	2^7
2Dsec	2^9	2^{10}	2^8	2^7
CMgauss	2^8	2^9	2^8	2^7
CMprev	2^8	2^8	2^8	2^7
AMgauss	2^8	2^9	2^9	2^7
AMprev	2^9	2^9	2^9	2^8
SVRG2	2^{10}	2^9	2^9	2^8
	mushrooms	phishing	w8a	rcv1
SVRG	2^8	2^6	2^8	2^{10}
2D	2^8	2^6	2^9	2^{14}
2Dsec	2^8	2^6	2^9	2^{13}
CMgauss	2^8	2^6	2^9	2^9
CMprev	2^7	2^6	2^9	2^{10}
AMgauss	2^7	2^7	2^9	2^9
AMprev	2^8	2^6	2^9	2^{10}
SVRG2	2^9	2^6	2^9	2^{10}

Table 1: Best empirical stepsize for each (problem, method) pair. In general, incorporating a Hessian approximation allows for larger stepsizes.

7 Conclusion

Control variates are at the core of stochastic variance-reducing methods. We proposed the first dynamic method which updates these control variates at every timestep, leading to faster empirical and theoretical convergence as well as increased robustness. While we applied it to SVRG, we hope it can be extended to other methods such as SAGA.

We also showed that the Hessian could be used beyond reconditioning and that efficient approximations existed. Though reconditioning in stochastic variance-reducing methods has been explored, it remains to be seen whether using the Hessian for both reconditioning and gradient tracking simultaneously is beneficial.

Finally, we believe there is still more to do to bridge the gap between stochastic and batch methods and we hope this work will open the way to further analyses of the importance of control variates in optimization.

Acknowledgements: RMG acknowledges the support of the FSMP postdoctoral fund. FB acknowledges support from the European Research Council (grant SEQUOIA 724063).

References

[1] Bruce Christianson. "Automatic Hessians by reverse accumulation". In: *IMA Journal of Numerical Analysis* 12.2 (1992), pp. 135–150.

[2] W C Davidon. "Variance algorithms for minimization". In: *Computer Journal* 10 (1968), pp. 406–410.

[3] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1646–1654.

[4] By R Fletcher and M J D Powell. "A Rapidly Convergent Descent Method for Minimization". In: *The Computer Journal* 6.2 (1963), pp. 163–168.

[5] Donald Goldfarb. "A Family of Variable-Metric Methods Derived by Variational Means". In: *Mathematics of Computation* 24.109 (1970), pp. 23–26.

[6] Robert M. Gower and Jacek Gondzio. "Action constrained quasi-Newton methods". In: *arXiv:1412.8045v1* (2014).

[7] Rie Johnson and Tong Zhang. "Accelerating stochastic gradient descent using predictive variance reduction". In: *Advances in Neural Information Processing Systems*. 2013, pp. 315–323.

[8] T Kern and A Gyorgy. "SVRG++ with non-uniform sampling". In: Neural Information Processing Systems Foundation, Inc., 2016.

[9] Nicolas Le Roux, Mark Schmidt, and Francis Bach. "A stochastic gradient method with an exponential convergence rate for finite training sets". In: *Advances in Neural Information Processing Systems*. 2012, pp. 2663–2671.

[10] Julien Mairal. "Optimization with First-Order Surrogate Functions". In: *Proceedings of The 30th International Conference on Machine Learning*. 2013, pp. 783–791.

[11] Jorge Nocedal. "Updating Quasi-Newton Matrices with Limited Storage". In: *Mathematics of Computation* 35 (July 1980), pp. 773–782.

[12] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[13] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*. John Wiley & Sons, 2016.

[14] Mark W. Schmidt et al. "Non-Uniform Stochastic Average Gradient Method for Training Conditional Random Fields". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*. 2015.

[15] Mark Schmidt. "Convergence rate of stochastic gradient with constant step size". In: (2014).

[16] Mark Schmidt, Nicolas Le Roux, and Francis Bach. "Minimizing Finite Sums with the Stochastic Average Gradient". In: *arXiv preprint arXiv:1309.2388* (2013).

[17] Shai Shalev-Shwartz and Tong Zhang. "Stochastic Dual Coordinate Ascent Methods for Regularized Loss". In: *Journal of Machine Learning Research* 14.1 (Feb. 2013), pp. 567–599. ISSN: 1532-4435.

[18] Andrea Walther. "Computing sparse Hessians with automatic differentiation". In: *ACM Trans. Math. Software* 34.1 (Jan. 2008), Art. 3, 15.

[19] Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. "Linear convergence with condition number independent access of full gradients". In: *Advances in Neural Information Processing Systems*. 2013, pp. 980–988.