Stochastic Block BFGS: Squeezing More Curvature out of Data

Robert M. Gower* Donald Goldfarb† Peter Richtárik‡

March 31, 2016

Abstract

We propose a novel limited-memory stochastic block BFGS update for incorporating enriched curvature information in stochastic approximation methods. In our method, the estimate of the inverse Hessian matrix that is maintained by it, is updated at each iteration using a sketch of the Hessian, i.e., a randomly generated compressed form of the Hessian. We propose several sketching strategies, present a new quasi-Newton method that uses stochastic block BFGS updates combined with the variance reduction approach SVRG to compute batch stochastic gradients, and prove linear convergence of the resulting method. Numerical tests on large-scale logistic regression problems reveal that our method is more robust and substantially outperforms current state-of-the-art methods.

1 Introduction

We design a new stochastic variable-metric (quasi-Newton) method—the stochastic block BFGS method—for solving the Empirical Risk Minimization (ERM) problem:

$$\min_{x\in\mathbb{R}^d} f(x) \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{1}$$

We assume the loss functions $f_i : \mathbb{R}^d \to \mathbb{R}$ to be convex and twice differentiable and focus on the setting where the number of *data points (examples)* (n) is very large.

To solve (1), we employ iterative methods of the form

$$x_{t+1} = x_t - \eta H_t g_t, \tag{2}$$

where $\eta > 0$ is a stepsize, $g_t \in \mathbb{R}^d$ is an estimate of the gradient $\nabla f(x_t)$ and $H_t \in \mathbb{R}^{d\times d}$ is a positive definite estimate of the inverse Hessian matrix, that is $H_t \approx \nabla^2 f(x_t)^{-1}$. We refer to H_t as the *metric matrix* [1].

*School of Mathematics, University of Edinburgh, gower.robert@gmail.com

†Department of Industrial Engineering and Operations Research, Columbia University, New York, goldfarb@columbia.edu

‡School of Mathematics, University of Edinburgh, peter.richtarik@ed.ac.uk. This author would like to acknowledge support from EPSRC Grant EP/K02325X/1 and EPSRC Fellowship EP/N005538/1.

[1] Methods of the form (2) can be seen as estimates of a gradient descent method under the metric defined by H_t. Indeed, let $\langle x, y\rangle_{H_t} \stackrel{\text{def}}{=} \langle H_t^{-1}x, y\rangle$ for any $x, y \in \mathbb{R}^d$ denote an inner product, then the gradient of $f(x_t)$ in this metric is $H_t\nabla f(x_t)$.

The most successful classical optimization methods fit the format (2), such as gradient descent ($H_t = I$), Newton's method $\left(H_t = \nabla^2 f(x_t)^{-1}\right)$, and the quasi-Newton methods $\left(H_t \approx \nabla^2 f(x_t)^{-1}\right)$; all with $g_t = \nabla f(x_t)$. The difficulty in our setting is that the large number of data points makes the computational costs of a single iteration of these classical methods prohibitively expensive.

To amortize these costs, the current state-of-the-art methods use subsampling, where g_t and H_t are calculated using only derivatives of the subsampled function

$$f_S(x) \stackrel{\text{def}}{=} \frac{1}{|S|} \sum_{i \in S} f_i(x),$$

where $S \subseteq [n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$ is a subset of examples selected uniformly at random. Using the subsampled gradient $\nabla f_S(x)$ as a proxy for the gradient is the basis for (minibatch) stochastic gradient descent (SGD), but also for many successful variance-reduced methods [32, 36, 18, 19, 5, 35, 20] that make use of the subsampled gradient in calculating g_t.

Recently, there has been an effort to calculate H_t using subsampled Hessian matrices $\nabla^2 f_T(x_t)$, where $T \subseteq [n]$ is sampled uniformly at random and independently of S [6, 31]. The major difficulty in this approach is that calculating $\nabla^2 f_T(x_t)$ can be computationally expensive and, when d is large, storing $\nabla^2 f_T(x_t)$ in memory can be infeasible. One fairly successful solution to these issues [2, 25] is to use a single Hessian-vector product $\nabla^2 f_T(x_t)v$, where $v \in \mathbb{R}^d$ is a suitably selected vector, to update H_t by the limited-memory (L-BFGS) [26] version of the classical BFGS [1, 7, 9, 37] method. Calculating this Hessian-vector product can be done inexpensively using the directional derivative

$$\nabla^2 f_T(x_t)v = \frac{d}{d\alpha} \nabla f_T(x_t + \alpha v)\Big|_{\alpha=0}. \tag{3}$$

In particular, when using automatic differentiation techniques [4, 16] or backpropagation on a neural network [27], evaluating the above costs at most five times as much as the cost of evaluating the subsampled gradient $\nabla f_T(x_t)$.

Using only a single Hessian-vector product to update H_t yields only a very limited amount of curvature information, and thus may result in an ineffective metric matrix. The block BFGS method addresses this issue. The starting point in the development of the block BFGS method is the simple observation that, ideally, we would like the metric matrix H_t to satisfy the inverse equation

$$H_t \nabla^2 f_T(x_t) = I,$$

since then H_t would be the inverse of an unbiased estimate of the Hessian. But solving the inverse equation is computationally expensive. So instead, we propose that H_t should instead satisfy a *sketched* version of this equation, namely

$$H_t \nabla^2 f_T(x_t) D_t = D_t, \tag{4}$$

where $D_t \in \mathbb{R}^{d \times q}$ is a randomly generated matrix which has relatively few columns ($q \ll d$). The *sketched subsampled Hessian* $\nabla^2 f_T(x_t) D_t$ can be calculated efficiently through q directional derivatives of the form (3).

Note that (4) has possibly an infinite number of solutions, including the inverse of $\nabla^2 f_T(x_t)$. To determine H_t uniquely, we maintain a previous estimate $H_{t-1} \in \mathbb{R}^{d \times d}$ and project H_{t-1} onto the space of symmetric matrices that satisfy (4). The resulting update, applied to H_{t-1} to arrive at H_t, is the block BFGS update.

In the remainder of the paper we describe the block BFGS update, propose a new limited-memory block BFGS update and introduce several new sketching strategies. We conclude by presenting the results of numerical tests of a method that combines the limited-memory block BFGS update with the SVRG/S2GD method [18, 19], and demonstrate that our new method yields dramatically better results when compared to SVRG, or the SVRG method coupled with the classical L-BFGS update as proposed in [25].

1.1 Contributions

This paper makes five main contributions:

(a) New metric learning framework. We develop a *stochastic block BFGS update*[2] for approximately tracking the inverse of the Hessian of f. This technique is novel in two ways: the update is more flexible than the traditional BFGS update as it works by employing the actions of a subsampled Hessian on a *set* of *random vectors* rather than just on a single deterministic vector, as is the case with the standard BFGS. That is, we use *block sketches* of the subsampled Hessian.

(b) Stochastic block BFGS method. Our block BFGS update is capable of incorporating *enriched* second-order information into gradient based stochastic approximation methods for solving problem (1). In this paper we illustrate the power of this strategy in conjunction with the strategy employed in the SVRG method of [18] for computing variance-reduced stochastic gradients. We prove that the resulting combined method is linearly convergent and empirically demonstrate its ability to substantially outperform current state-of-the-art methods.

(c) Limited-memory method. To make the stochastic block BFGS method applicable to large-scale problems, we devise a new limited-memory variant of it. As is the case for L-BFGS [26], our limited-memory approach allows for a user-defined amount of memory to be set aside. But unlike L-BFGS, our limited-memory approach allows one to use the available memory to store more recent curvature information, encoded in sketches of previous Hessian matrices. This development of a new limited block BFGS method should also be of general interest to the optimization community.

(d) Factored form. We develop a limited-memory factored form of the block BFGS update. While factorized versions of standard quasi-Newton methods were proposed in the 1970's (e.g., see [8, 10]), as far as we know, no efficient limited memory versions of such methods have been developed. Factored forms are important as they can be used to enforce positive definiteness of the metric even in the presence numerical imprecision. Furthermore, we use the factored form in calculating new sketching matrices.

(e) Adaptive sketching. Not only can sketching be used to tackle the large dimensions of the Hessian, but it can also simultaneously precondition the inverse equation (4). We present a self-conditioning (i.e., adaptive) sketching that makes use of the efficient factored form of the block BFGS method developed earlier. We also present a sketching approach based on using previous

[2]In this paper we use the word "update" to denote metric learning, i.e., an algorithm for updating one positive definite matrix into another.

search directions. Our numerical tests show that adaptive sketching can in practice lead to a significant speedup in comparison with sketching from a fixed distribution.

1.2 Background and Related Work

The first stochastic variable-metric method developed that makes use of subsampling was the online L-BFGS method [34]. In this work the authors adapt the L-BFGS method to make use of subsampled gradients, among other empirically verified improvements. The regularized BFGS method [24, 23] also makes use of stochastic gradients, and further modifies the BFGS update by adding a regularizer to the metric matrix.

The first method to use subsampled Hessian-vector products in the BFGS update, as opposed to using differences of stochastic gradients, was the SQN method [2]. Recently, [25] propose combining SQN with SVRG. The resulting method performs very well in numerical tests. In our work we combine a novel *stochastic block BFGS* update with SVRG, and prove linear convergence. The resulting method is more versatile and superior in practice to SQN as it can capture more useful curvature information.

The update formula, that we refer to as the block BFGS update, has a rather interesting background. The formula first appeared in 1983 in unpublished work by Schnabel [33] on designing quasi-Newton methods that make use of multiple secant equations. Schnabel's method requires several modifications that stem from the lack of symmetry and positive definiteness of the resulting update. Later, and completely independently, the block BFGS update appears in the domain decomposition literature [22] as a preconditioner, where it is referred to as the balancing preconditioner. In that work, the motivation and derivation are very different from those used in the quasi-Newton literature; for instance, no variational interpretation is given for the method. The balancing preconditioner was subsequently taken out of the PDE context and tested as a general purpose preconditioner for solving a single linear system and systems with a changing right hand side [15]. Furthermore, in [15] the authors present a factored form of the update in a different context, which we adapt for limited-memory implementation. Finally, and again independently, a family of block quasi-Newton methods that includes the block BFGS is presented in [13] through a variational formulation and in [17] using Bayesian inference.

2 Stochastic Block BFGS Update

The stochastic block BFGS update, applied to H_{t-1} to obtain H_t, is defined by the projection

$$\begin{aligned} H_t = \arg\min_{H\in\mathbb{R}^{d\times d}} & \|H - H_{t-1}\|_t^2 \\ \text{subject to } & H\nabla^2 f_T(x_t)D_t = D_t,\ H = H^T, \end{aligned} \tag{5}$$

where

$$\|H\|_t^2 \overset{\text{def}}{=} \mathbf{Tr}\left(H\nabla^2 f_T(x_t)H^T\nabla^2 f_T(x_t)\right),$$

and $\mathbf{Tr}(\cdot)$ denotes the trace. The method is stochastic since $D_t \in \mathbb{R}^{d\times q}$ is a random matrix. The constraint in (5) serves as a *fidelity* term, enforcing that H_t be a symmetric matrix that satisfies a sketch of the inverse equation (4). The objective in (5) acts as a *regularizer*, and ensures that the difference between H_t and H_{t-1} is a low rank update. The choice of the objective is special yet in another sense: recent results show that if the iteration (5) is applied to a fixed invertible matrix A

in place of the subsampled Hessian, then the matrices H_t converges to the inverse A^{-1} at a linear rate [13]. This is yet one more reason to expect that in our setting the matrices H_t approximately track the inverse Hessian. For similar techniques applied to the problem of solving linear systems, we refer to [12, 14].

The solution to (5) is

$$H_t \;=\; D_t\Delta_t D_t^T + \left(I - D_t\Delta_t Y_t^T\right) H_{t-1}\left(I - Y_t\Delta_t D_t\right), \tag{6}$$

where $\Delta_t \overset{\text{def}}{=} (D_t^T Y_t)^{-1}$ and $Y_t \overset{\text{def}}{=} \nabla^2 f_T(x_t) D_t$. This solution was given in [11, 17] and in [33] for multiple secant equations.

Note that stochastic block BFGS (6) yields the same matrix H_t if D_t is replaced by any other matrix $\tilde{D}_t \in \mathbb{R}^{d\times q}$ such that $\mathbf{span}(\tilde{D}_t) = \mathbf{span}(D_t)$. It should also be pointed out that the matrix H_t produced by (6) is not what would be generated by a sequence of q rank-two BFGS updates based on the q columns of D_t unless the columns of D_t are $\nabla^2 f_T(x_t)$-conjugate. Note that these columns would be conjugate if they were generated by the BFGS method applied to the problem of minimizing a strictly convex quadratic function with Hessian $\nabla^2 f_T(x_t)$, using exact line-search.

We take this opportunity to point out that stochastic block BFGS can generate the metric used in the Stochastic Dual Newton Ascent (SDNA) method [29] as a special case. Indeed, when $H_{t-1} = 0$, then H_t is given by

$$H_t \;=\; D_t(D_t^T\nabla^2 f_T(x_t)D_t)^{-1}D_t^T. \tag{7}$$

When the sketching matrix D_t is a random column submatrix of the identity, then (7) is the positive semidefinite matrix used in calculating the iterates of the SDNA method [29]. However, SDNA operates in the dual of (1).

3 Stochastic Block BFGS Method

The goal of this paper is to design a method that uses a low-variance estimate of the gradient, but also gradually incorporates *enriched* curvature information. To this end, we propose to combine the stochastic variance reduced gradient (SVRG) approach [18] with our *novel stochastic block BFGS update,* described in the previous section. The resulting method is Algorithm 1.

Algorithm 1 has an outer loop in k and an inner loop in t. In the outer loop, the *outer* iterate $w_k \in \mathbb{R}^d$ and the full gradient $\nabla f(w_k)$ are computed. In the inner loop, both the estimate of the gradient g_t and our metric H_t are updated using the SVRG update and the block BFGS update, respectively.

To form the sketching matrix D_t we employ one of the following three strategies:

(a) Gaussian sketch. D_t has standard Gaussian entries sampled i.i.d at each iteration.

(b) Previous search directions delayed. Let us write $d_t = -H_t g_t$ for the search direction used in step t of the method. In this strategy we store L such search directions as columns of matrix D_t: $D_t = [d_{t+1-L}, \ldots, d_t]$, and then update H_t only once every L inner iterations.

(c) Self-conditioning. We sample $C_t \subseteq [d]$ uniformly at random and set $D_t = L_{t-1}I_{:C_t} = [L_{t-1}]_{:C_t}$, where $L_{t-1}L_{t-1}^T = H_{t-1}$ and $I_{:C_t}$ denotes the concatenation of the columns of the identity matrix indexed by a set $C_t \subset [d]$. Thus the sketching matrix is formed with a random subset of columns of a factored form of H_t. The idea behind this strategy is that the ideal sketching matrix should be $D_t = (\nabla^2 f_T(x_t))^{-1/2}$ so that the sketch not only compresses but also acts as a preconditioner on the inverse equation (4). It was also shown in [13] that this choice of sketching matrix can accelerate the convergence of H_t to the inverse of a fixed matrix. In Section 3.2 we describe in detail how to efficiently maintain and update the factored form.

Algorithm 1 Stochastic Block BFGS Method

inputs: $w_0 \in \mathbb{R}^d$, stepsize $\eta > 0$, s = subsample size, q = sample action size, m = size of the inner loop.
initiate: $H_{-1} = I$ ($d \times d$ identity matrix)
for $k = 0, 1, 2, \ldots$ **do**
  Compute the full gradient $\mu = \nabla f(w_k)$
  Set $x_0 = w_k$
  for $t = 0, \ldots, m-1$ **do**
    Sample $S_t, T_t \subseteq [n]$, independently
    Compute a variance-reduced stochastic gradient: $g_t = \nabla f_{S_t}(x_t) - \nabla f_{S_t}(x_0) + \mu$
    Form $D_t \in \mathbb{R}^{d \times q}$ so that $\text{rank}(D_t) = q$
    Compute $Y_t = \nabla^2 f_{T_t}(x_t) D_t$ (without ever forming the $d \times d$ matrix $\nabla^2 f_{T_t}(x_t)$)
    Compute ${D_t}^T Y_t$ and its Cholesky factorization (this implicitly forms $\Delta_t = (D_t^T Y_t)^{-1}$)
    Option I: Use (6) to obtain H_t and set $d_t = -H_t g_t$
    Option II: Compute $d_t = -H_t g_t$ via Algorithm 2
    Set $x_{t+1} = x_t + \eta d_t$
  end for
  Option I: Set $w_{k+1} = x_m$
  Option II: Set $w_{k+1} = x_i$, where i is selected uniformly at random from $[m] = \{1, 2, \ldots, m\}$
end for
output: w_{k+1}

3.1 Limited-Memory Block BFGS

When d is large, we cannot store the $d \times d$ matrix H_t. Instead, we store M block triples, consisting of previous block curvature pairs and the inverse of their products

$$(D_{t+1-M}, Y_{t+1-M}, \Delta_{t+1-M}), \ldots, (D_t, Y_t, \Delta_t). \tag{8}$$

With these triples we can form the H_t operator implicitly by using a block limited-memory two loop recurrence. To describe this two loop recurrence, let $V_t \overset{\text{def}}{=} I - D_t \Delta_t Y_t^T$.

The block BFGS update (6) with memory parameter M can be expanded as a function of the M block triples (8) and of H_{t-M} as

$$\begin{aligned} H_t &= V_t H_{t-1} V_t^T + D_t \Delta_t D_t^T \\ &= V_t \cdots V_{t+1-M} H_{t-M} V_{t+1-M}^T \cdots V_t^T + \sum_{i=t}^{t+1-M} V_t \cdots V_{i+1} D_i \Delta_i D_i^T V_{i+1}^T \cdots V_t^T. \end{aligned}$$

Since we do not store H_t for any t, we do not have access to H_{t-M}. In our experiments we simply set $H_{t-M} = I$ (the identity matrix). Other, more sophisticated, choices are possible, but we do not explore them further here. Using the above expansion, the action of the operator H_t on a vector v can be efficiently calculated using Algorithm 2.

Algorithm 2 Block L-BFGS Update (Two-loop Recursion)

inputs: $g_t \in \mathbb{R}^d$, D_i, $Y_i \in \mathbb{R}^{d\times q}$ and $\Delta_i \in \mathbb{R}^{q\times q}$ for $i \in \{t+1-M, \dots, t\}$.
initiate: $v = g_t$
for $i = t, \dots, t-M+1$ **do**
  $\alpha_i = \Delta_i D_i^T v$
  $v = v - Y_i \alpha_i$
end for
for $i = t-M+1, \dots, t$ **do**
  $\beta_i = \Delta_i Y_i^T v$
  $v = v + D_i(\alpha_i - \beta_i)$
end for
output $H_t g_t = v$

The total cost in floating point operations of executing Algorithm 2 is $Mq(4d + 2q)$. In our experiments $M = 5$ and q will be orders of magnitude less than d, typically $q \leq \sqrt{d}$. Thus the cost of applying Algorithm 2 is approximately $O(d^{3/2})$. This does not include the cost of computing the product ${D_t}^T Y_t$ ($O(q^2 d)$ operations) and its Cholesky factorization ($O(q^3)$ operations), which is done outside of Algorithm 2. The two places in Algorithm 2 where multiplication by Δ_i is indicated is in practice performed by solving two triangular systems using the Cholesky factor of ${D_i}^T Y_i$. We do this because it is more numerically stable than explicitly calculating the inverse matrix Δ_i.

3.2 Factored Form

Here we develop a new efficient method for maintaining and updating a *factored form of the metric matrix*. This facilitates the development of a novel idea which we call *self-conditioning sketch*.

Let $L_{t-1} \in \mathbb{R}^{d\times d}$ be invertible such that $L_{t-1}L_{t-1}^T = H_{t-1}$. Further, let $G_t = (D_t^T L_{t-1}^{-T} L_{t-1}^{-1} D_t)^{1/2}$ and $R_t = \Delta_t^{1/2}$. An update formula for the factored form of H_t, i.e., for L_t for which $H_t = L_t L_t^T$, was recently given (in a different context) in [15]:

$$L_t = V_t L_{t-1} + D_t R_t G_t^{-1} D_t^T L_{t-1}^{-T}. \tag{9}$$

This factored form of H_t is too costly to compute because it requires inverting L_{t-1}. However, if we let $D_t = L_{t-1} I_{:C_t}$, where $C_t \subset [d]$, then (9) reduces to

$$L_t = V_t L_{t-1} + D_t R_t I_{C_t:} \tag{10}$$

which can be computed efficiently. Furthermore, this update of the factored form (10) is amenable to recursion and can thus be expanded as

$$\begin{aligned} L_t &= V_t \left(V_{t-1}L_{t-2} + D_{t-1}R_{t-1}I_{C_{t-1}:}\right) + D_t R_t I_{C_t:} \\ &= V_t \cdots V_{t+2-M} V_{t+1-M} L_{t-M} + V_t \cdots V_{t+2-M} D_{t+1-M} R_{t+1-M} I_{C_{t+1-M}:} + D_t R_t I_{C_t:} \quad . \end{aligned} \tag{11}$$

By storing M previous curvature pairs (8) and additionally the sets $C_{t+1-M}, \ldots, C_t$, we can calculate the action of L_t on a matrix $V \in \mathbb{R}^{d\times q}$ by using (11), see Algorithm 3. To the best of our knowledge, this is the first limited-memory factored form in the literature. Since we do not store L_t explicitly for any t we do not have access to L_{t-M}, required in computing (11). Thus we simply use $L_{t-M} = I$.

Algorithm 3 Block L-BFGS Update (Factored loop recursion for computing L_tV)

inputs: V, $D_i, Y_i, \Delta_i \in \mathbb{R}^{d\times q}$ and $C_i \subset [d]$, for $i \in \{t+1-M, \ldots, t\}$.
initiate: $W = V$
for $i = t+1-M, \ldots, t$ **do**
  $W = W - D_i\Delta_i Y_i^T W + D_i R_i W_{C_i:}$
end for
output W (we will have $W = L_tV$)

Again, we can implement a more numerically stable version of Algorithm 3 by storing the Cholesky factor of $D_i^T Y_i$ and using triangular solves, as opposed to calculating the inverse matrix $\Delta_i = (D_i^T Y_i)^{-1}$.

4 Convergence

In this section we prove that Algorithm 1 converges linearly. Our analysis relies on the following assumption, and is a combination of novel insights and techniques from [19] and [25].

Assumption 1. *There exist constants $0 < \lambda \leq \Lambda$ such that*

$$\lambda I \preceq \nabla^2 f_T(x) \preceq \Lambda I \tag{12}$$

for all $x \in \mathbb{R}^d$ and all $T \subseteq [n]$.

We need two technical lemmas, whose proofs are given in Sections 7 and 8 at the end of the paper.

Lemma 1. *Let H_t be the result of applying the limited-memory Block BFGS update with memory M, as implicitly defined by Algorithm 2. Then there exists positive constants $\Gamma \geq \gamma > 0$ such that for all t we have*

$$\gamma I \preceq H_t \preceq \Gamma I. \tag{13}$$

A proof of this lemma is given in Appendix 2, where in particular it is shown that the lower bound satisfies $\gamma \geq \frac{1}{1+M\Lambda}$ and the upper bound satisfies

$$\Gamma \leq (1 + \sqrt{\kappa})^{2M}\left(1 + \frac{1}{\lambda(2\sqrt{\kappa}+\kappa)}\right), \tag{14}$$

where $\kappa \overset{\text{def}}{=} \Lambda/\lambda$.

We now state a bound on the norm of the SVRG variance-reduced gradient for minibatches.

Lemma 2. *Suppose Assumption 1 holds, let w_* be the unique minimizer of f and let $w, x \in \mathbb{R}^d$. Let $\mu = \nabla f(w)$ and $g = \nabla f_S(x) - \nabla f_S(w) + \mu$. Taking expectation with respect to S, we have*

$$\mathbb{E}\left[\|g\|_2^2\right] \leq 4\Lambda(f(x) - f(w_*)) + 4(\Lambda - \lambda)(f(w) - f(w_*)). \tag{15}$$

The following theorem guarantees the linear convergence of Algorithm 1.

Theorem 1. *Suppose that Assumption 1 holds. Let w_* be the unique minimizer of f. When Option II is used in Algorithm 1, we have for all $k \geq 0$ that*

$$\mathbb{E}\left[f(w_k) - f(w_*)\right] \leq \rho^k \mathbb{E}\left[f(w_0) - f(w_*)\right],$$

where the convergence rate is given by

$$\rho = \frac{1/2m\eta + \eta\Gamma^2\Lambda(\Lambda - \lambda)}{\gamma\lambda - \eta\Gamma^2\Lambda^2} < 1,$$

assuming we have chosen $\eta < \gamma\lambda/(2\Gamma^2\Lambda^2)$ and that we choose m large enough to satisfy[3]

$$m \geq \frac{1}{2\eta\left(\gamma\lambda - \eta\Gamma^2\Lambda(2\Lambda - \lambda)\right)}.$$

Proof. Since $g_t = \nabla f_{S_t}(x_t) - \nabla f_{S_t}(w_k) + \mu$ and $x_{t+1} = x_t - \eta H_t g_t$ in Algorithm 1, from (12) we have that

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) + \eta\nabla f(x_t)^T d_t + \frac{\eta^2\Lambda}{2}\|d_t\|_2^2 \\ &= f(x_t) - \eta\nabla f(x_t)^T H_t g_t + \frac{\eta^2\Lambda}{2}\|H_t g_t\|_2^2. \end{aligned}$$

Taking expectation conditioned on x_t (i.e., with respect to S_t, T_t and D_t) and using Lemma 1 we have

$$\begin{aligned} \mathbb{E}\left[f(x_{t+1}) \,|\, x_t\right] &\leq f(x_t) - \eta\mathbb{E}\left[\nabla f(x_t)^T H_t \nabla f(x_t) \,|\, x_t\right] + \frac{\eta^2\Lambda}{2}\mathbb{E}\left[\|H_t g_t\|_2^2 \,|\, x_t\right] \\ &\overset{\text{Lemma 1}}{\leq} f(x_t) - \eta\gamma\|\nabla f(x_t)\|_2^2 + \frac{\eta^2\Gamma^2\Lambda}{2}\mathbb{E}\left[\|g_t\|_2^2 \,|\, x_t\right]. \end{aligned} \tag{16}$$

Introducing the notation $\delta_f(x) \overset{\text{def}}{=} f(x) - f(w_*)$ and applying Lemma 2 and the fact that strongly convex functions satisfy the inequality $\|\nabla f(x)\|_2^2 \geq 2\lambda\delta_f(x)$ for all $x \in \mathbb{R}^d$, gives

$$\begin{aligned} \mathbb{E}\left[f(x_{t+1}) \,|\, x_t\right] &\overset{(16)+(15)}{\leq} f(x_t) - 2\eta\gamma\lambda\delta_f(x_t) + 2\eta^2\Gamma^2\Lambda\left(\Lambda\delta_f(x_t)) + (\Lambda - \lambda)\delta_f(w_k)\right)\right) \\ &= f(x_t) - \alpha\delta_f(x_t) + \beta\delta_f(w_k). \end{aligned}$$

where $\alpha = 2\eta\left(\gamma\lambda - \eta\Gamma^2\Lambda^2\right)$ and $\beta = 2\eta^2\Gamma^2\Lambda(\Lambda - \lambda)$. Taking expectation, summing over $t = 0, \ldots, m-1$ and using telescopic cancellation gives

$$\begin{aligned} \mathbb{E}\left[f(x_m)\right] &= \mathbb{E}\left[f(x_0)\right] - \alpha\left(\sum_{t=1}^{m-1}\mathbb{E}\left[\delta_f(x_t)\right]\right) + m\beta\mathbb{E}\left[\delta_f(w_k)\right] \\ &= \mathbb{E}\left[f(w_k)\right] - m\alpha\mathbb{E}\left[\delta_f(w_{k+1})\right] + m\beta\mathbb{E}\left[\delta_f(w_k)\right], \end{aligned}$$

[3] By our assumption on η, the expression on the right is nonnegative.

where we used that $w_k = x_0$ and $\sum_{t=1}^{m} \mathbb{E}[x_t] = m\mathbb{E}[w_{k+1}]$ which is a consequence of using Option II in Algorithm 1. Rearranging the above gives

$$\begin{aligned} 0 &\leq \mathbb{E}[f(w_k) - f(x_m)] - m\alpha\mathbb{E}[\delta_f(w_{k+1})] + m\beta\mathbb{E}[\delta_f(w_k)] \\ &\leq \mathbb{E}[\delta_f(w_k)] - m\alpha\mathbb{E}[\delta_f(w_{k+1})] + m\beta\mathbb{E}[\delta_f(w_k)] \\ &= -m\alpha\mathbb{E}[\delta_f(w_{k+1})] + (1 + m\beta)\mathbb{E}[\delta_f(w_k)] \end{aligned}$$

where we used that $f(w_*) \leq f(x_m)$. Using that $\eta < \gamma\lambda/(2\Gamma^2\Lambda^2)$, it follows that

$$\mathbb{E}[\delta_f(w_{k+1})] \leq \frac{1 + 2m\eta^2\Gamma^2\Lambda(\Lambda - \lambda)}{2m\eta(\gamma\lambda - \eta\Gamma^2\Lambda^2)}\mathbb{E}[\delta_f(w_k)]. \qquad \square$$

5 Numerical Experiments

To validate our approach, we compared our algorithm to SVRG [18] and the variable-metric adaption of SVRG presented in [25], which we refer to as the `MNJ` method. We tested the methods on seven empirical risk minimization problems with a logistic loss and L2 regularizer, that is, we solved

$$\min_w \sum_{i=1}^{n} \ln\left(1 + \exp(-y_i\langle a^i, w\rangle)\right) + \frac{1}{n}\|w\|_2^2, \tag{17}$$

where $A = [a^1, \ldots, a^n] \in \mathbb{R}^{d\times n}$ and $y \in \{0,1\}^n$ are the given data. We have also employed the standard "bias trick"[4]. For our experiments we used data from the LIBSVM collection [3]. All the methods were implemented in MATLAB. All the code for the experiments can be downloaded from `http://www.maths.ed.ac.uk/~prichtar/i_software.html`.

We tested three variants of Algorithm 1, each specified by the use of a different sketching matrix. In Table 1 we present a key to the abbreviations used in all our figures. The first three methods, `gauss`, `prev` and `fact`, are implementations of the three variants (a), (b) and (c), respectively, of Algorithm 1 using the three different sketching methods discussed at the start of Section 3. In these three methods the M stands for the number of stored curvature pairs (8) used.

For each method and a given parameter choice, we tried the stepsizes

$$\eta \in \{10^0,\ 5\cdot 10^{-1},\ 10^{-1},\ \ldots,\ 10^{-7},\ 5\cdot 10^{-8},\ 10^{-8}\}$$

and reported the one that gave the best results. We used the error $f(x_t) - f(w_*)$ for the y-axis in all our figures[5]. We calculated $f(w_*)$ by running all methods for 30 passes over the data, and then taking the minimum function value.

Finally, we used $m = \lfloor n/|S_t| \rfloor$ for the number of inner iterations in all variants of Algorithm 1, SVRG and `MNJ`, so that all methods perform an entire pass over the data before recalculating the gradient.

[4]The bias trick is to add an additional *bias* variable $\beta \in \mathbb{R}$ so that the exponent in (17) is $-y_i(\langle a^i, w\rangle + \beta)$. This is done efficiently by simple concatenating a row of ones to data matrix so that $\langle [a^i\ 1], [w\ \beta]\rangle = \langle a^i, w\rangle + \beta$, for $i = 1, \ldots, n$.

[5]Thanks to Mark Schmidt, whose code `prettyPlot` was used to generate all figures: `https://www.cs.ubc.ca/~schmidtm/Software/prettyPlot.html`. Note that in all plots the markers (circles, plus signs, triangles ... etc) are used to help distinguish the different methods, and are not related to the iteration count.

Method	Description
`gauss_`q`_`M	$D_t \in \mathbb{R}^{d\times q}$ with i.i.d Gaussian entries
`prev_`L`_`M	$D_t = [d_t, \ldots, d_{t-L+1}]$. Updated every L inner iterations
`fact_`q`_`M	$D_t = L_{t-1}I_{:C}$ where $C \subset [n]$ sampled uniformly at random and $\lvert C\rvert = q$
`MNJ_`$\lvert T_t\rvert$	Algorithm 1 in [25] where $\lvert T_t\rvert$ = size of Hessian subsampling and $L = 10$

Table 1: A key to the abbreviations used for each method

5.1 Parameter investigation

In our first set of tests we explored the parameter space of the `prev` variant of Algorithm 1. By fixing all parameters but one, we can see how sensitive the `prev` method is to the singled-out parameter, but also, build some intuition as to what value or, interval of values, yields the best results for this parameter. We focused our tests on the `prev` method since it proved to be overall, the most robust method.

In Figure 1 (a) we depict the results of varying the memory parameter M, while fixing the remaining parameters. In particular, we fixed $|T_t| = |S_t| = 15$. In both the subplots of error $\times$ datapasses and error $\times$ time in Figure 1 (a), we see that $1 \leq M \leq 4$ resulted in the best performance. Furthermore, the error $\times$ datapasses is insensitive to increasing the memory, since increasing the memory parameter does not incur in any additional data passes. On larger dimensional problems we found that approximately $M = 5$ yielded the overall best performance. Thus we used $M = 5$ in our tests on large-scale problems in Sections 5.2 and 5.3.

In Figure 1 (b) we experimented with varying $|T_t|$, the size of the Hessian subsampling. Here the results of the error $\times$ datapasses subplot conflict with those of the the error $\times$ time subplots. While in the error $\times$ time subplot the method improves as $|T_t|$ increases, in the error $\times$ datapasses subplot the method improves as $|T_t|$ decreases. As a compromise, in Sections 5.2 and 5.3 we use $|S_t| = |T_t|$ as our default choice.

In Figure 2 (a) we experimented with varying the size of gradient subsampling and Hessian subsampling jointly with $|S_t| = |T_t|$. In both the error $\times$ datapasses and the error $\times$ time subplot the range

$$\lceil n^{1/3} \rceil = 32 \leq |S_t| = |T_t| \leq 724 = \lceil n^{1/2} \rceil,$$

resulted in a good performance. Based on this experiment, we use $|S_t| = |T_t| = n^{1/2}$ as our default choice in Sections 5.2 and 5.3. Note that when $|S_t|$ is large the method passes through the data in fewer iterations and consequentially less time. This is why the method terminated early (in time) when $|S_t|$ is large.

Finally in Figure 2 (b) we vary the parameter L, that is, the number of previous search directions used to form the columns of D_t. From these tests we can see that using a value that is too small, e.g. $L = 1$, or a value that is too large $L = 2\lceil\sqrt{d}\rceil = 24$, results in an inefficient method in terms of both time and datapasses. Instead, we get the best performance when $\lceil d^{1/4} \rceil \leq L \leq \lceil d^{1/2} \rceil$. Thus on the first five problems we used either $L = d^{1/3}$ or $L = d^{1/4}$, depending on which gave the best performance. As for the last two problems: `rcv1-train.binary` and `url-combined`, we found that

$L = d^{1/4}$ was too large. Instead, we probed the set $L \in \{2, \ldots, 10\}$ for an L that resulted in a reasonable performance.

Through these four experiments in Figures 1 and 2, we can conclude that the `prev` method is not overly sensitive to the choice of these parameters. That is, the method works well for a range of parameter choices. This is in contrast with choosing the stepsize parameter, whose choice can make the difference between a divergent method and a fast method.

5.2 Data passes

We now compare all the methods in Table 1 in terms of error $\times$ datapasses. Since our experiment in Figure 1 (b) indicated that $|S_t| = |T_t|$ resulted in a reasonable method, for simplicity, we used the same subsampling for the gradient and Hessian in all of our methods; that is $S_t = T_t$. This is not necessarily an optimal choice. Furthermore, we set the subsampling size $|S_t| = \sqrt{n}$.

For the `MNJ` method, we used the suggestion of the authors of both [2] and [25], and chose $L = 10$ and $|T_t| \approx L|S_t|$ so that the computational workload performed by L inner iterations of the SVRG method was approximately equal to that of applying the L-BFGS metric once. The exact rule we found to be efficient was

$$|T_t| = \left\lfloor \min \left\{ \frac{L|S_t|}{2}, n^{2/3} \right\} \right\rfloor.$$

We set the memory to 10 for the `MNJ` method in all tests, which is a standard choice for the L-BFGS method.

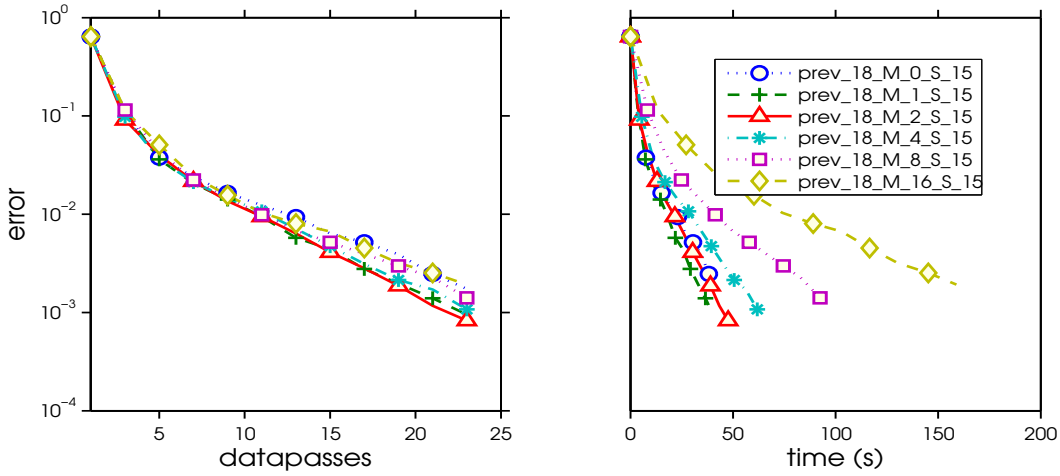
On the problems with d significantly smaller then n, such as Figures 1(c) and 1(d), all the methods that make use of curvature information performed similarly and significantly better than the SVRG method. The `prev` method proved to be the best overall and the most robust, performing comparably well on problems with $d \ll n$, such as Figures 1(c) and 1(d), but was also the most efficient method in Figures 1(a), 1(b), 1(e) (shared being most efficient with `MNJ`) and Figure 2(a). The only problem on which the `prev` method was not the most efficient method was on the `url-combined` problem in Figure 2(b), where the `MNJ` method proved to be the most efficient.

Overall, these experiments illustrate that incorporating curvature information results in a fast and robust method. Moreover, the added flexibility of the block BFGS update to incorporate more curvature information, as compared to a single Hessian-vector product in the `MNJ` method, can significantly improve the convergence of the method, as can be seen in Figures 1(a) and 1(b).

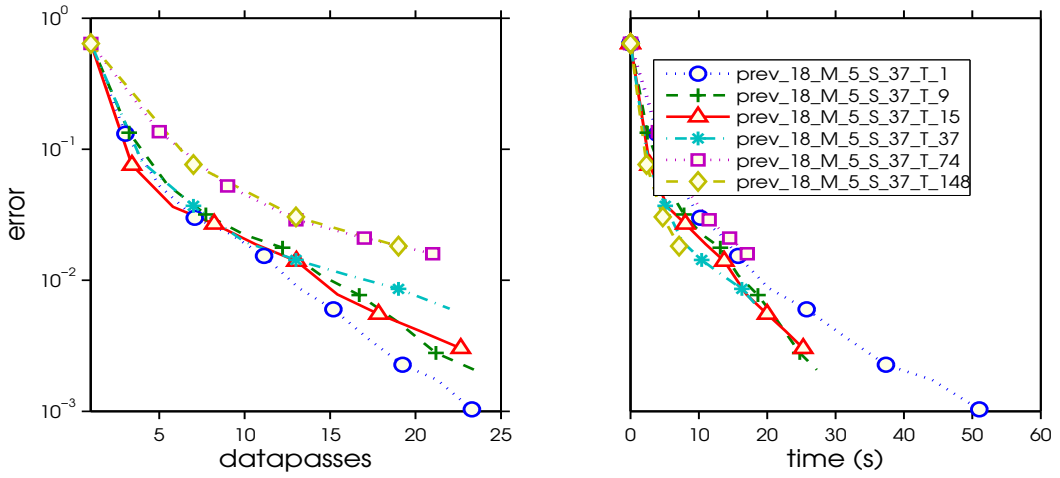
5.3 Timed

In Figures 5 and 6, we compare the evolution of the error over time for each method. While measuring time is implementation and machine dependent, we include these time plots as to provide further insight into the methods performance. Note that we did not use any sophisticated implementation tricks such as "lazy" gradient updates [19], but instead implemented each method as originally designed so that the methods can be compared on a equal footing.

The results in these tests corroborate with our conclusions in Section 5.2

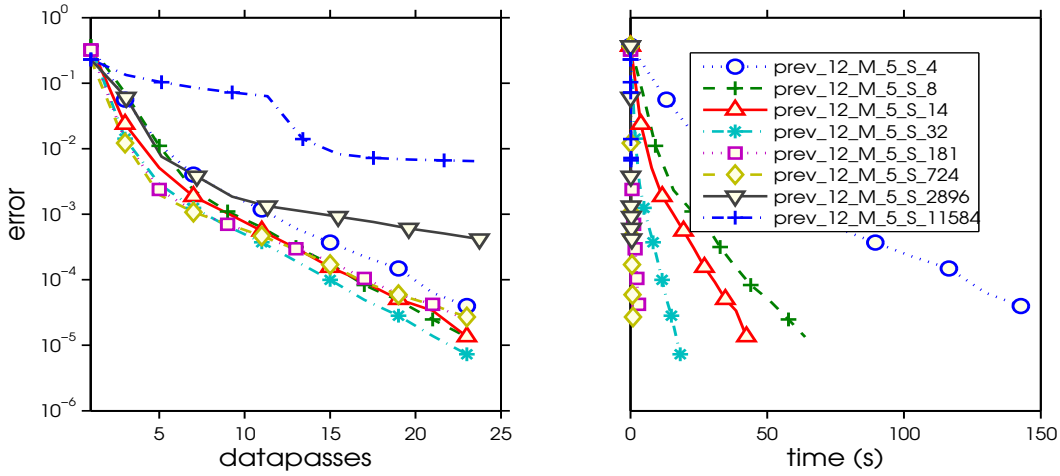


(a) memory M

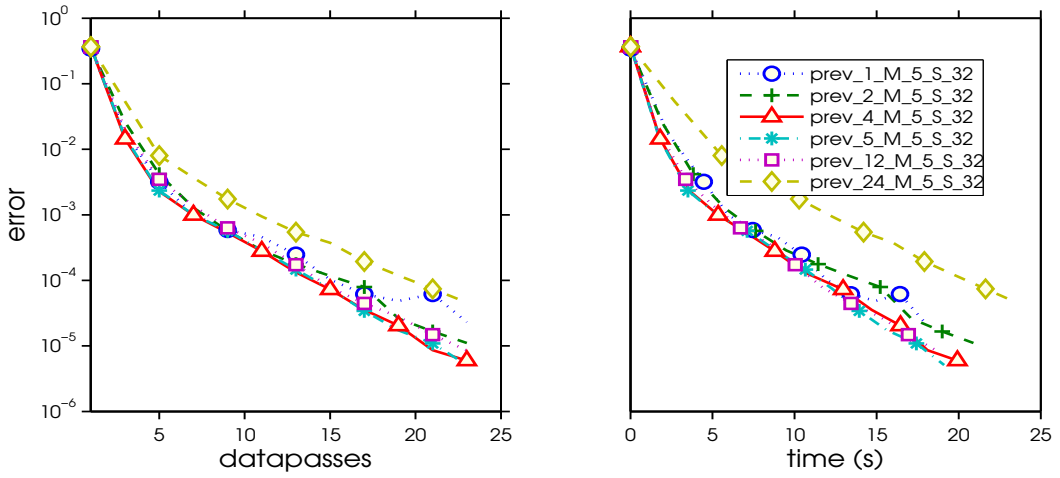


(b) Hessian subsampling $|T_t|$

Figure 1: `w8a` $(d; n) = (300;\ 49,749)$

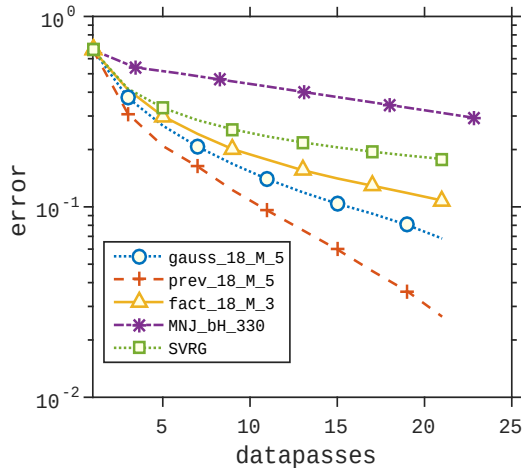


(a) subsampling $|S_t| = |T_t|$

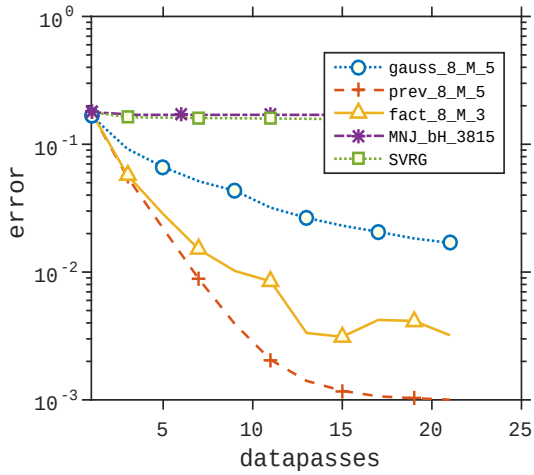


(b) update size L

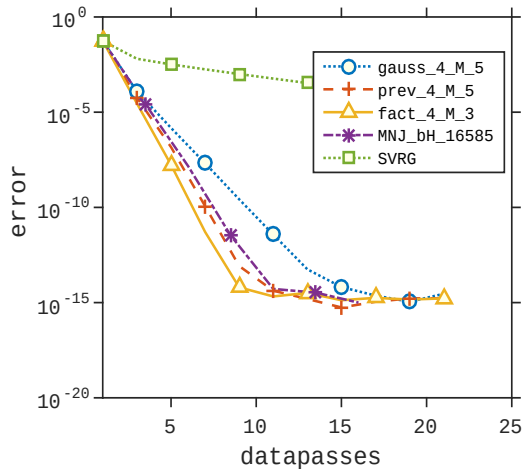
Figure 2: a9a $(d; n) = (123;\ 32,561)$



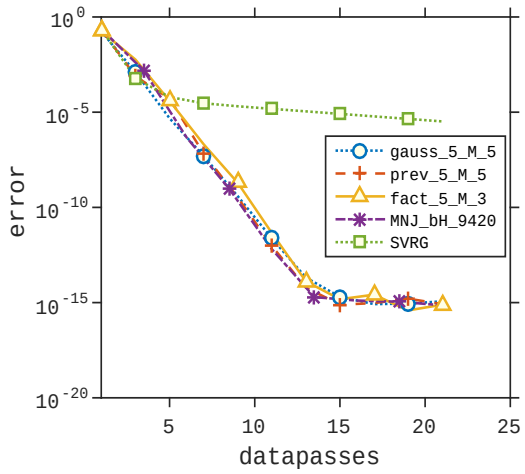
(a) `gissette_scale`



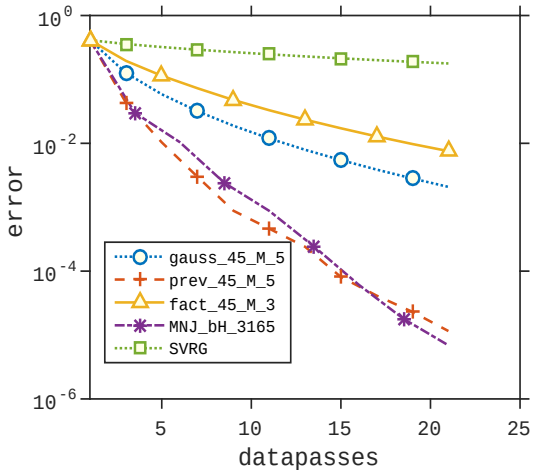
(b) `covtype-libsvm-binary`



(c) `HIGGS`

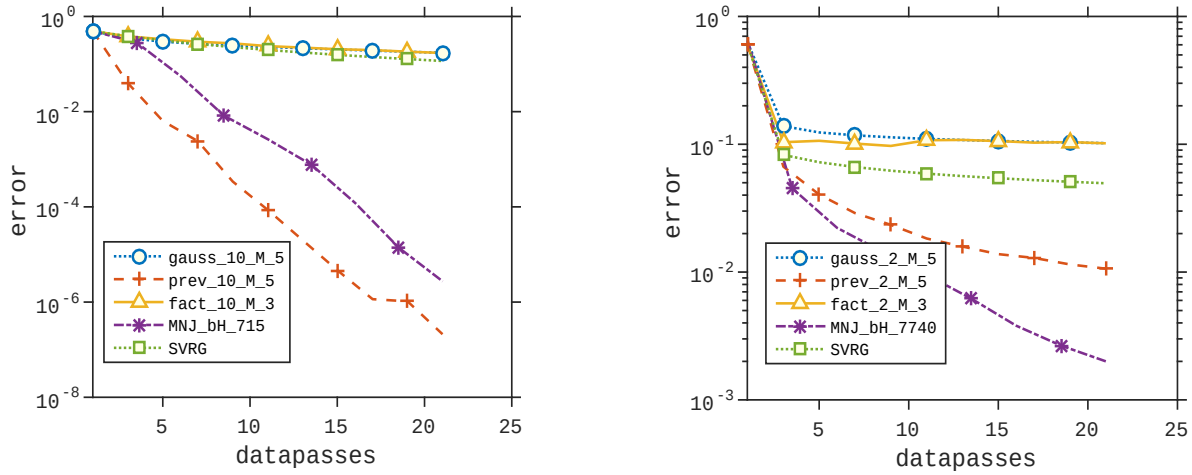


(d) `SUSY`



(e) `epsilon_normalized`

Figure 3: (a) `gissette_scale` $(d; n) = (5,001;\ 6,000)$ (b) `covtype-libsvm-binary` $(d; n) = (55;\ 581,012)$ (c) `HIGGS` $(d; n) = (29;\ 11,000,000)$ (d) `SUSY` $(d; n) = (19;\ 3,548,466)$ (e) `epsilon_normalized` $(d; n) = (2,001;\ 400,000)$



(a) `rcv1-train.binary` (b) `url-combined`

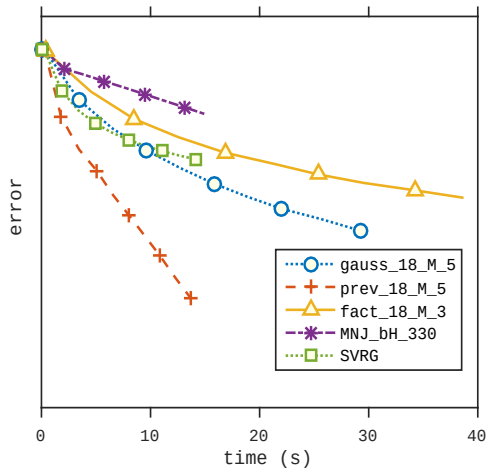
Figure 4: (a) `rcv1-train.binary` $(d; n) = (47,237;\ 20,242)$ (b) `url-combined` $(d; n) = (3,231,962;\ 2,396,130)$.

6 Extensions

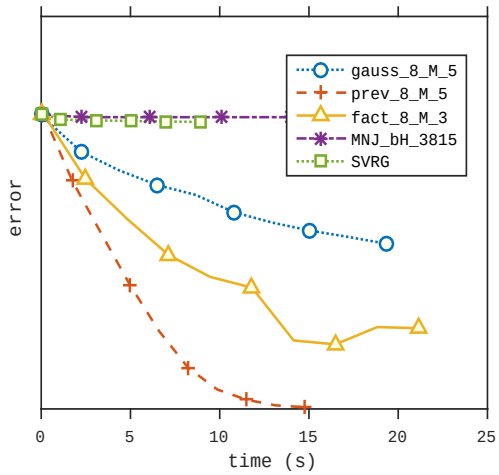
This work opens up many new research avenues. For instance, sketching techniques are increasingly successful tools in large scale machine learning, numerical linear algebra, optimization and computer science. Therefore, one could employ a number of new sketching methods in the block BFGS method, such as the Walsh-Hadamard matrix [28, 21]. Using new sophisticated sketching methods, combined with the block BFGS update, could result in even more efficient and accurate estimates of the underlying curvature.

Viewed in terms of sketching, the `MNJ` method [25] and our Stochastic Block BFGS method with `prev` sketching follow opposite strategies. While the `MNJ` method sketches the Hessian matrix with a single vector v, where v is the average over a combination of previous search directions (a very coarse approximation), our `prev` variant uses all previous search directions to form the sketching matrix D_t (a finer approximation). Deciding between these two extremes or how to combine them could be done adaptively by examining the curvature matrix $D_t \nabla f(x_t) D_t$. When the previous search directions are almost collinear, $D_t \nabla f(x_t) D_t$ becomes ill-conditioned. In this case one should form matrix D_t with less columns using fewer or a coarser combination of previous search directions, while if $D_t \nabla f(x_t) D_t$ is well-conditioned, one could use more or a finer combination of previous search directions.

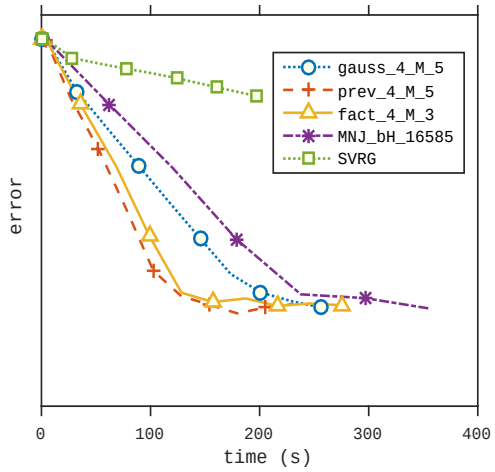
While in this work we have for simplicity focused on utilizing our metric learning techniques in conjunction with SVRG, they can be used with other optimization algorithms as well, including SGD [30], SDCA [36] and more.



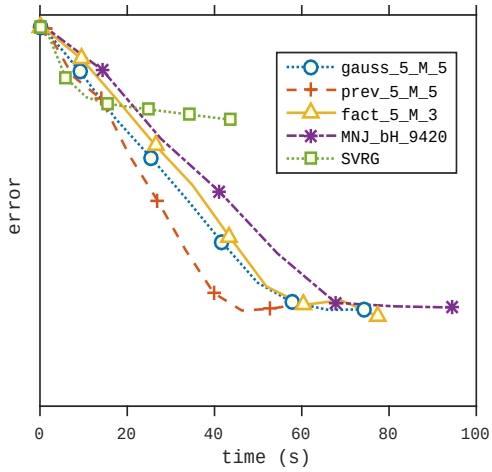
(a) `gissette_scale`



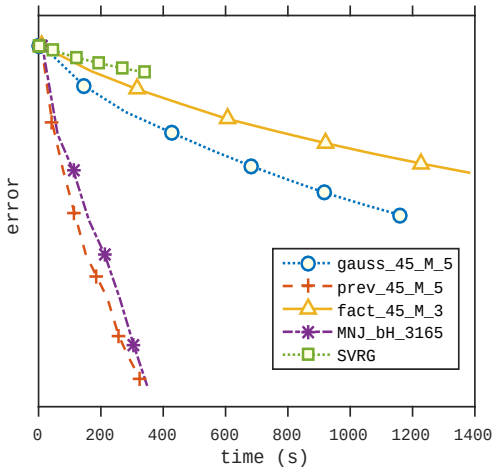
(b) `covtype-libsvm-binary`



(c) `HIGGS`

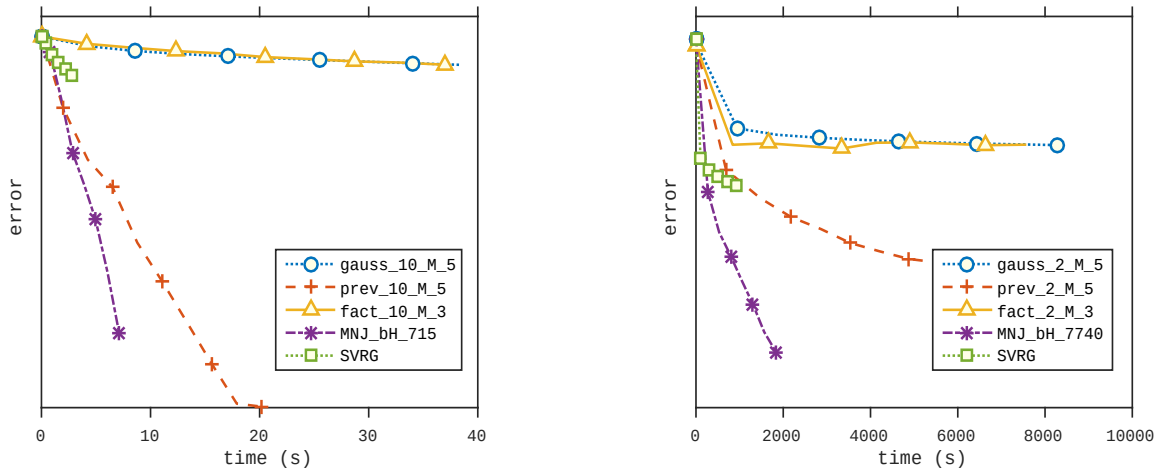


(d) `SUSY`



(e) `epsilon_normalized`

Figure 5: (a) `gissette_scale` $(d; n) = (5,001;\ 6,000)$ (b) `covtype-libsvm-binary` $(d; n) = (55;\ 581,012)$ (c) `HIGGS` $(d; n) = (29;\ 11,000,000)$ (d) `SUSY` $(d; n) = (19;\ 3,548,466)$ (e) `epsilon_normalized` $(d; n) = (2,001;\ 400,000)$



(a) `rcv1-train.binary` (b) `url-combined`

Figure 6: (a) `rcv1-train.binary` $(d; n) = (47,237;\ 20,242)$ (b) `url-combined` $(d; n) = (3,231,962;\ 2,396,130)$.

7 Proof of Lemma 2

Let $h_S(w) = f_S(w) - f_S(w_*) - \nabla f_S(w_*)^T(w - w_*)$. Note that $h_S(w)$ achieves its minimum at w_* and $h_S(w_*) = 0$. Furthermore, $\nabla^2 h_S(w) \preceq \Lambda I$ from Assumption 1. Consequently, for every $b \in \mathbb{R}^d$ we have

$$0 = h_S(w_*) \leq h_S(w + b) \leq h_S(w) + \nabla h_S(w)^T b + \frac{\Lambda}{2} \|b\|_2^2 .$$

Minimizing the right hand side of the above in b gives

$$0 = h_S(w_*) \leq h_S(w) - \frac{1}{2\Lambda} \|\nabla h_S(w)\|_2^2 .$$

Re-arranging the above and switching back to f, we have

$$\|\nabla f_S(w) - \nabla f_S(w_*)\|_2^2 \leq 2\Lambda \left(f_S(w) - f_S(w_*) - \nabla f_S(w_*)^T(w - w_*) \right).$$

Recalling the notation $\delta_f(x) \stackrel{\text{def}}{=} f(x) - f(w_*)$ and taking expectation with respect to S gives

$$\mathbb{E}\left[\|\nabla f_S(w) - \nabla f_S(w_*)\|_2^2 \right] \leq 2\Lambda \delta_f(w). \tag{18}$$

Now we apply the above to obtain

$$\begin{aligned} \mathbb{E}\left[\|g\|_2^2 \right] &\leq 2\mathbb{E}\left[\|\nabla f_S(x) - \nabla f_S(w_*)\|_2^2 \right] + 2\mathbb{E}\left[\|\nabla f_S(w) - \nabla f_S(w_*) - \mu\|_2^2 \right] \\ &\leq 2\mathbb{E}\left[\|\nabla f_S(x) - \nabla f_S(w_*)\|_2^2 \right] + 2\mathbb{E}\left[\|\nabla f_S(w) - \nabla f_S(w_*)\|_2^2 \right] - 2\|\nabla f(w)\|_2^2 . \end{aligned}$$

where we used $\|a+b\|_2^2 \leq 2\|a\|_2^2 + 2\|b\|_2^2$ in the first inequality and $\mu = \mathbb{E}\left[\nabla f_S(w_k) - \nabla f_S(w_*)\right]$. Finally,

$$\begin{aligned}\mathbb{E}\left[\|g\|_2^2\right] &\overset{(18)}{\leq} 4\Lambda\left(\delta_f(x) + \delta_f(w)\right) - 2\|\nabla f(w)\|_2^2 \\ &\leq 4\Lambda\delta_f(x) + 4(\Lambda - \lambda)\delta_f(w).\end{aligned}$$

In the last inequality we used the fact that strongly convex functions satisfy $\|\nabla f(x)\|_2^2 \geq 2\lambda\,\delta_f(x)$ for all $x \in \mathbb{R}^d$.

8 Proof of Lemma 1

To simplify notation, we define $G \overset{\text{def}}{=} \nabla^2 f_T(x_t)$, $\Delta = \Delta_t$, $Y = Y_t$, $H = H_{t-1}$, $H^+ = H_t$, $B = H^{-1}$, $B^+ = (H^+)^{-1}$ and $V = Y\Delta D^T$. Thus, the block BFGS update can be written as

$$H^+ = H - V^T H - HV + V^T HV + D\Delta D^T.$$

Proposition 2.2 in [11] proves that so long as G and H (and hence B) are positive definite and D has full rank, then H^+ is positive definite and non-singular, and consequently, B^+ is well defined and positive definite. Using the Sherman-Morrison-Woodbury identity the update formula for B^+, as shown in the Appendix in [11], is given by

$$B^+ = B + Y\Delta Y^T - BD(D^T BD)^{-1}D^T B. \tag{19}$$

We will now bound $\lambda_{\max}(H^+) = \|H^+\|_2$ from above and $\lambda_{\min}(H^+) = 1/\|B^+\|_2$ from below.

Let $C = BD(D^T BD)^{-1}D^T B$. Then since $C \succeq 0$, $B - C \preceq B$ and hence, $\|B - C\|_2 \leq \|B\|$ and

$$\|B^+\|_2 \leq \|B\|_2 + \|Y\Delta Y^T\|_2.$$

Now, letting $G^{\frac{1}{2}}$ and $G^{-\frac{1}{2}}$ denote the unique square root of G and its inverse, and defining $U = G^{\frac{1}{2}}D$, we have

$$D\Delta D^T = G^{-\frac{1}{2}}U(U^T U)^{-1}U^T G^{-\frac{1}{2}} = G^{-\frac{1}{2}}PG^{-\frac{1}{2}},$$

where $P = U(U^T U)^{-1}U^T$ is an orthogonal projection matrix. Moreover, it is easy to see that

$$Y\Delta Y^T = G^{\frac{1}{2}}PG^{\frac{1}{2}} \quad \text{and} \quad V = Y\Delta D^T = G^{\frac{1}{2}}PG^{-\frac{1}{2}}.$$

Since $\|MN\|_2 \leq \|M\|_2\|N\|_2$ and $\|P\|_2 = 1$, we have $\|D\Delta D^T\|_2 \leq \|G^{-1}\|_2$, $\|Y\Delta Y^T\|_2 \leq \|G\|_2$ and $\|Y\Delta D^T\|_2 \leq \|G^{-\frac{1}{2}}\|_2\|G^{\frac{1}{2}}\|_2$.

Hence,

$$\|B^+\|_2 \leq \|B\|_2 + \|G\|_2 \overset{(12)}{\leq} \|B\|_2 + \Lambda. \tag{20}$$

Furthermore,

$$\begin{aligned}\|H^+\|_2 &\leq \|H\|_2 + 2\|H\|_2\|G^{-\frac{1}{2}}\|_2\|G^{\frac{1}{2}}\|_2 + \|H\|_2\|G^{-1}\|_2\|G\|_2 + \|G^{-1}\|_2 \\ &\leq (1 + 2\sqrt{\kappa} + \kappa)\|H\|_2 + \frac{1}{\lambda} \qquad (21) \\ &= \alpha\|H\|_2 + \frac{1}{\lambda}, \qquad (22)\end{aligned}$$

where $\kappa = \Lambda/\lambda$ and $\alpha = (1+\sqrt{\kappa})^2$.

Since we use a memory of M block triples (D_i, Y_i, Δ_i), and the metric matrix H_t is the result of applying, at most, M block updates BFGS (6) to H_0, we have that

$$\lambda_{\max}(B_t) = \|B_t\| \overset{(20)}{\leq} \|B_{t-M}\| + M\Lambda, \tag{23}$$

and hence that

$$\gamma = \lambda_{\min}(H_t) \overset{(23)}{\geq} \frac{1}{\|B_{t-M}\| + M\Lambda}.$$

Finally, since $\alpha = (1+\sqrt{\kappa})^2$, we have

$$\begin{aligned}
\Gamma &= \lambda_{\max}(H_t) \\
&= \|H_t\| \\
&\overset{(22)}{\leq} \alpha^M \|H_{t-M}\| + \frac{1}{\lambda} \sum_{i=0}^{M-1} \alpha^i \\
&= \alpha^M \|H_{t-M}\| + \frac{1}{\lambda} \frac{\alpha^M - 1}{\alpha - 1} \\
&\leq (1+\sqrt{\kappa})^{2M} \left(\|H_{t-M}\| + \frac{1}{\lambda(2\sqrt{\kappa}+\kappa)} \right).
\end{aligned}$$

The bound (14) now follows by observing that $H_{t-M} = I$.

References

[1] C. G. Broyden. "Quasi-Newton methods and their application to function minimisation". *Mathematics of Computation* 21.99 (1967), pp. 368–381.

[2] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. "A stochastic quasi-Newton method for large-scale optimization". *arXiv:1401.7020v2* (2015).

[3] C. C. Chang and C. J. Lin. "LIBSVM : a library for support vector machines". *ACM Transactions on Intelligent Systems and Technology* 2.3 (Apr. 2011), pp. 1–27.

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

[5] A. Defazio, F. Bach, and S. Lacoste-Julien. "SAGA: a fast incremental gradient method with support for non-strongly convex composite objectives". *arXiv:1407.0202* (2014).

[6] M. A. Erdogdu and A. Montanari. "Convergence rates of sub-sampled Newton methods". *arXiv:1508.02810* (2015).

[7] R. Fletcher. "A new approach to variable metric algorithms". *The Computer Journal* 13.3 (1970), pp. 317–323.

[8] P. E. Gill and W. Murray. "Quasi-Newton methods for unconstrained optimization". *IMA Journal of Applied Mathematics* 9.1 (1972), pp. 91–108.

[9] D. Goldfarb. "A family of variable-metric methods derived by variational means". *Mathematics of Computation* 24.109 (1970), pp. 23–26.

[10] D. Goldfarb. "Factorized variable metric methods for unconstrained optimization". *Mathematics of Computation* 30.136 (1976), pp. 796–811.

[11] R. M. Gower and J Gondzio. "Action constrained quasi-Newton methods". *arXiv:1412.8045v1* (2014).

[12] R. M. Gower and P. Richtárik. "Randomized iterative methods for linear systems". *SIAM Journal on Matrix Analysis and Applications* 36.4 (2015), pp. 1660–1690.

[13] R. M. Gower and P. Richtárik. "Randomized quasi-Newton updates are linearly convergent matrix inversion algorithms". *arXiv:1602.01768* (2016).

[14] R. M. Gower and P. Richtárik. "Stochastic dual ascent for solving linear systems". *arXiv:1512.06890* (2015).

[15] S. Gratton, A. Sartenaer, and J. T. Ilunga. "On a class of limited memory preconditioners for large-scale nonlinear least-squares problems". *SIAM Journal on Optimization* 21.3 (2011), pp. 912–935.

[16] A. Griewank and A. Walther. *Evaluating Derivatives*. Second Edi. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2008.

[17] P. Hennig. "Probabilistic interpretation of linear solvers". *SIAM Journal on Optimization* 25.1 (2015), pp. 234–260.

[18] R. Johnson and T. Zhang. "Accelerating stochastic gradient descent using predictive variance reduction". In: *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 315–323.

[19] J. Konečný and P. Richtárik. "S2GD: semi-stochastic gradient descent methods". *arXiv:1312.1666* (2013).

[20] J. Konečný, J. Liu, P. Richtárik, and M. Takáč. "Mini-batch semi-stochastic gradient descent in the proximal setting". *IEEE Journal of Selected Topics in Signal Processing* 10.2 (2016), pp. 242–255.

[21] Y. Lu, P. Dhillon, D. P. Foster, and L. Ungar. "Faster ridge regression via the subsampled randomized Hadamard transform". In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 369–377.

[22] J. Mandel and M. Brezina. "Balancing domain decomposition: theory and performance in two and three dimensions". *Communications in Numerical Methods in Engineering* 9.3 (1993), pp. 233–241.

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

[24] A. Mokhtari and A. Ribeiro. "Regularized stochastic BFGS algorithm". *IEEE Transactions on Signal Processing* 62 (23 2014), pp. 1109–1112.

[25] P. Moritz, R. Nishihara, and M. I. Jordan. "A linearly-convergent stochastic L-BFGS algorithm". *arXiv:1508.02087v1* (2015).

[26] J. Nocedal. "Updating quasi-Newton matrices with limited storage". *Mathematics of Computation* 35.151 (1980), p. 773.

[27] B. A. Pearlmutter. "Fast exact multiplication by the Hessian". *Neural Computation* 6.1 (1994), pp. 147–160.

[28] M. Pilanci and M. J. Wainwright. "Iterative Hessian sketch: Fast and accurate solution approximation for constrained least-squares". *to appear in Journal of Machine Learning Research* (2015), pp. 1–33.

[29] Z. Qu, P. Richtárik, M. Takáč, and O. Fercoq. "SDNA: Stochastic dual Newton ascent for empirical risk minimization". *arXiv:1502.02268v1* (2015).

[30] H. Robbins and S. Monro. "A stochastic approximation method". *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[31] F. Roosta-Khorasani and M. W. Mahoney. "Sub-sampled Newton methods I: globally convergent algorithms". *arXiv:1601.04737* (2016).

[32] M. Schmidt, N. Le Roux, and F. Bach. "Minimizing finite sums with the stochastic average gradient". *arXiv:1309.2388* (2013).

[33] R. B. Schnabel. *Quasi-Newton Methods Using Multiple Secant Equations ; CU-CS-247-83.* Tech. rep. Computer Science Technical Reports. Paper 244, 1983.

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

[35] S. Shalev-Shwartz and T. Zhang. "Accelerated mini-batch stochastic dual coordinate ascent". In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 378–385.

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

[37] D. F. Shanno. "Conditioning of quasi-Newton methods for function minimization". *Mathematics of Computation* 24.111 (1971), pp. 647–656.