SIAM J. OPTIM.
Vol. 26, No. 2, pp. 1008–1031

A STOCHASTIC QUASI-NEWTON METHOD FOR LARGE-SCALE OPTIMIZATION[*]

R. H. BYRD[†], S. L. HANSEN[‡], JORGE NOCEDAL[§], AND Y. SINGER[¶]

Abstract. The question of how to incorporate curvature information into stochastic approximation methods is challenging. The direct application of classical quasi-Newton updating techniques for deterministic optimization leads to noisy curvature estimates that have harmful effects on the robustness of the iteration. In this paper, we propose a stochastic quasi-Newton method that is efficient, robust, and scalable. It employs the classical BFGS update formula in its limited memory form, and is based on the observation that it is beneficial to collect curvature information pointwise, and at spaced intervals. One way to do this is through (subsampled) Hessian-vector products. This technique differs from the classical approach that would compute differences of gradients at every iteration, and where controlling the quality of the curvature estimates can be difficult. We present numerical results on problems arising in machine learning that suggest that the proposed method shows much promise.

Key words. stochastic optimization, quasi-Newton, sub sampling, large scale optimization

AMS subject classifications. 65K05, 90C06, 90C30, 90C55

DOI. 10.1137/140954362

1. Introduction.

In many applications of machine learning, one constructs very large models from massive amounts of training data. Learning such models imposes high computational and memory demands on the optimization algorithms employed to learn the models. In some applications, a full-batch (sample average approximation) approach is feasible and appropriate. However, in most large-scale learning problems, it is imperative to employ stochastic approximation algorithms that update the prediction model based on a relatively small subset of the training data. These algorithms are particularly suited for settings where data are perpetually streamed to the learning process; examples include computer network traffic, web search, online advertisement, and sensor networks.

The goal of this paper is to propose a quasi-Newton method that operates in the stochastic approximation regime. We employ the well-known limited memory BFGS updating formula, and show how to collect second-order information that is reliable enough to produce stable and productive Hessian approximations. The key is to compute average curvature estimates using a sufficiently large sample, and at spaced intervals so as to amortize their cost. A convenient way to do so is by using

[*] Received by the editors January 27, 2014; accepted for publication (in revised form) February 9, 2016; published electronically April 14, 2016.
http://www.siam.org/journals/siopt/26-2/95436.html

[†] Department of Computer Science, University of Colorado, Boulder, CO 80309 (richard.byrd@colorado.edu). This author was supported by National Science Foundation grant DMS-1216554 and Department of Energy grant DE-SC0001774.

[‡] Department of Engineering Sciences and Applied Mathematics, Northwestern University, Evanston, IL 60202 (samanthahansen2014@u.northwestern.edu). This author was supported by National Science Foundation grant DMS-0810213.

[§] Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208 (nocedal@eecs.northwestern.edu). This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number FG02-87ER25047. This author was also supported by the Office of Naval Research award N000141410313.

[¶] Google Research (singer@google.com).

(subsampled) Hessian-vector products. This ensures sample uniformity and avoids the potentially harmful effects of differencing noisy gradients.

The problem under consideration is the minimization of a convex stochastic function,

$$\min_{w\in\mathbb{R}^n} F(w) = \mathbb{E}[f(w;\xi)], \tag{1.1}$$

where ξ is a random variable. Although problem (1.1) arises in other settings, such as simulation optimization [2], we assume for concreteness that ξ is a random instance consisting of an input-output pair (x,z). The vector x is typically referred to in machine learning as the input representation and z as the target output. In this setting, f typically takes the form

$$f(w;\xi) = f(w;x_i,z_i) = \ell(h(w;x_i);z_i), \tag{1.2}$$

where ℓ is a loss function into $\mathbb{R}_+$, and h is a prediction model parametrized by w. The collection of input-output pairs $\{(x_i,z_i)\}$, $i=1,\dots,N$, is referred to as the training set. The objective function (1.1) is defined using the empirical expectation

$$F(w) = \frac{1}{N}\sum_{i=1}^{N} f(w;x_i,z_i). \tag{1.3}$$

In learning applications with very large amounts of training data, it is common to use a minibatch stochastic gradient based on $b \overset{\triangle}{=} |\mathcal{S}| \ll N$ input-output instances, yielding the following estimate,

$$\nabla F_{\mathcal{S}}(w) = \frac{1}{b}\sum_{i\in\mathcal{S}} \nabla f(w;x_i,z_i). \tag{1.4}$$

The subset $\mathcal{S} \subset \{1,2,\dots,N\}$ is randomly chosen, with b sufficiently small so that the algorithm operates in the stochastic approximation regime. Therefore, the stochastic estimates of the gradient are substantially faster to compute than a gradient based on the entire training set.

Our optimization method employs iterations of the form

$$w^{k+1} = w^k - \alpha^k B_k^{-1}\nabla F_{\mathcal{S}}(w^k), \tag{1.5}$$

where B_k is a symmetric positive definite approximation to the Hessian matrix $\nabla^2 F(w)$, and $\alpha^k > 0$. Since the stochastic gradient is not an accurate approximation to the gradient of (1.3) it is essential (to guarantee convergence) that the step length parameter $\alpha^k \to 0$. In our experiments and analysis, α^k has the form $\alpha^k = \beta/k$, where $\beta > 0$ is given, but other choices can be employed.

A critical question is how to construct the Hessian approximation in a stable and efficient manner. For the algorithm to be scalable, it must update the inverse matrix $H_k = B_k^{-1}$ directly, so that (1.5) can be implemented as

$$w^{k+1} = w^k - \alpha^k H_k \nabla F_{\mathcal{S}}(w^k). \tag{1.6}$$

Furthermore, this step computation should require only $O(n)$ operations, as in limited memory quasi-Newton methods for deterministic optimization.

If we set $H_k = I$ and $\alpha^k = \beta/k$ in (1.6), we recover the classical Robbins–Monro method [22], which is also called the *stochastic gradient descent (SGD) method*. Under a strong convexity assumption, the number of iterations needed by this method to compute a function value that is within ϵ of the optimal value is $n\nu\kappa^2/\epsilon$, where κ is the condition number of the Hessian at the optimal solution, $\nabla^2 F(w^*)$, and ν is a parameter that depends on both the Hessian matrix and the gradient covariance matrix; see [15, 5]. Therefore, the SGD method is adversely affected by ill conditioning in the Hessian. In contrast, it is shown by Bottou and LeCun [6] that letting $H_k \to \nabla^2 F(w^*)^{-1}$ (at any rate) completely removes the dependency on κ from the complexity estimate. Although choosing H_k in such a manner is not viable in practice, it suggests that an appropriate choice of H_k may result in an algorithm that improves upon the SGD method.

In the next section, we present a stochastic quasi-Newton (SQN) method of the form (1.6) that is designed for large-scale applications. It employs the limited memory BFGS update, which is defined in terms of correction pairs (s, y) that provide an estimate of the curvature of the objective function $F(w)$ along the most recently generated directions. We propose an efficient way of defining these correction pairs that yields curvature estimates that are not corrupted by the effect of differencing the noise in the gradients. Our numerical experiments using problems arising in machine learning suggest that the new method is robust and efficient.

The paper is organized into 6 sections. The new algorithm is presented in section 2, and its convergence properties on strongly convex functions are discussed in section 3. Numerical experiments that illustrate the practical performance of the algorithm are reported in section 4. A literature survey on related stochastic quasi-Newton methods is given in section 5. The paper concludes in section 6 with some remarks about the contributions of the paper.

Notation. The terms Robbins–Monro method, stochastic approximation method, and SGD method are used in the literature to denote (essentially) the same algorithm. The first term is common in statistics, the second term is popular in the stochastic programming literature, and the acronym SGD is standard in machine learning. We will use the SGD in the discussion that follows.

2. A stochastic quasi-Newton method.

The success of quasi-Newton methods for deterministic optimization lies in the fact that they construct curvature information during the course of the optimization process, and this information is good enough to endow the iteration with a superlinear rate of convergence. In the classical BFGS method [10] for minimizing a deterministic function $F(w)$, the new inverse approximation H_{k+1} is uniquely determined by the previous approximation H_k and the correction pairs

$$y_k = \nabla F(w^{k+1}) - \nabla F(w^k), \quad s_k = w^{k+1} - w^k.$$

Specifically,

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \quad \text{with} \quad \rho_k = \frac{1}{y_k^T s_k}.$$

This BFGS update is well defined as long as the curvature condition $y_k^T s_k > 0$ is satisfied, which is always the case when $F(w)$ is strictly convex.

For large scale applications, it is necessary to employ a limited memory variant that is scalable in the number of variables, but enjoys only a linear rate of convergence.

This so-called L-BFGS method [18] is considered generally superior to the steepest descent method for deterministic optimization: it produces well-scaled and productive search directions that yield an approximate solution in fewer iterations and function evaluations.

When extending the concept of limited memory quasi-Newton updating to the stochastic approximation regime it is not advisable to mimic the classical approach for deterministic optimization and update the model based on information from only one iteration. This is because quasi-Newton updating is inherently an overwriting process rather than an averaging process, and therefore the vector y must reflect the action of the Hessian of the entire objective F given in (1.1)—something that is not achieved by differencing stochastic gradients (1.4) based on small samples.

We propose that an effective approach to achieving stable Hessian approximation is to *decouple* the stochastic gradient and curvature estimate calculations. Doing so provides the opportunity to use a different sample subset for defining y and the flexibility to add new curvature estimates at regular intervals instead of at each iteration. In order to emphasize that the curvature estimates are updated at a different schedule than the gradients, we use the subscript t to denote the number of times a new (s, y) pair has been calculated; this differs from the superscript k which counts the number of gradient calculations and variables updates.

The displacement s can be computed based on a collection of average iterates. Assuming that new curvature estimates are calculated every L iterations, we define s_t as the difference of disjoint averages between the $2L$ most recent iterations:

$$s_t = \bar{w}_t - \bar{w}_{t-1}, \quad \text{where} \quad \bar{w}_t = \sum_{i=k-L}^{k} w^i \tag{2.1}$$

(and $\bar{w}_{t-1}$ is defined similarly). In order to avoid the potential harmful effects of gradient differencing when $\|s_t\|$ is small, we chose to compute y_t via a Hessian vector product,

$$y_t = \nabla^2 F_{\mathcal{S}_H}(\bar{w}_t) s_t, \tag{2.2}$$

i.e., by approximating differences in gradients via a first-order Taylor expansion, where $\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)$ is a subsampled Hessian defined as follows. Let $\mathcal{S}_H \subset \{1, \ldots, N\}$ be a randomly chosen subset of the training examples and let

$$\nabla^2 F_{\mathcal{S}_H}(w) \overset{\triangle}{=} \frac{1}{b_H} \sum_{i \in \mathcal{S}_H} \nabla^2 f(w; x_i, z_i), \tag{2.3}$$

where b_H is the cardinality of $\mathcal{S}_H$.

We emphasize that the matrix $\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)$ is never constructed explicitly when computing y_t in (2.2), rather, the Hessian-vector product can be coded directly. To provide useful curvature information, $\mathcal{S}_H$ should be relatively large (see section 4), regardless of the size of b.

The pseudocode of the complete method is given in Algorithm 1.

The use of the Hessian-vector products (2.2) is convenient but one might be able to achieve the same goals using differences in gradients, i.e.,

$$\bar{y} = \nabla F_{\mathcal{S}}(\bar{w}_t) - \nabla F_{\mathcal{S}}(\bar{w}_{t-1}). \tag{2.4}$$

This would require, however, that the evaluation of these gradients employ the same sample $\mathcal{S}$ so as to obtain sample uniformity, which doubles the number of gradient evaluations, as well as the development of a strategy to prevent close gradient differences from magnifying round-off noise. In comparison, the use of Hessian-vector products takes care of these issues automatically, but it requires code for Hessian-vector computations (a task that is often not onerous).

Algorithm 1 SQN Method

Input: initial parameters w^1, positive integers M, L, step-length sequence $\alpha^k > 0$, gradient and Hessian samples sizes b and b_H

```
 1: Set $t = -1$                                    ▷ Records number of correction pairs computed
 2: for  $k = 1, \ldots,$ do
 3:     Choose a sample $\mathcal{S} \subset \{1, 2, \ldots, N\}$, with $|\mathcal{S}| = b$
 4:     Calculate stochastic gradient $\nabla F_{\mathcal{S}}(w^k)$ as defined in (1.4)
 5:     if  $t < 1$ then
 6:         $w^{k+1} = w^k - \alpha^k \nabla F_{\mathcal{S}}(w^k)$            ▷ Stochastic gradient iteration
 7:     else
 8:         $w^{k+1} = w^k - \alpha^k H_t \nabla F_{\mathcal{S}}(w^k)$, where $H_t$ is defined by Algorithm 2
 9:     end if
10:     if $\mathrm{mod}(k, L) = 0$ then               ▷ Compute correction pairs every $L$ iterations
11:         $t = t + 1$
12:         $\bar{w}_t = \sum_{j=k-L+1}^{k} w_j / L$
13:         if  $t > 0$ then
14:             Choose $\mathcal{S}_H \subset \{1, \ldots, N\}$, $|\mathcal{S}_H| = b_H$, to define $\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)$ by (2.3)
15:             Set $s_t = (\bar{w}_t - \bar{w}_{t-1}),\ y_t = \nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)(\bar{w}_t - \bar{w}_{t-1})$    ▷ correction pairs
16:         end if
17:     end if
18: end for
```

The L-BFGS step computation in step 8 follows standard practice [18]. Having chosen a memory parameter M, the matrix H_t is defined as the result of applying M BFGS updates to an initial matrix using the M most recent correction pairs $\{s_j, y_j\}_{j=t-M+1}^{t}$ computed by the algorithm. This procedure is mathematically described as follows.

Algorithm 2 Hessian Updating

Input: Updating counter t, memory parameter M, and correction pairs (s_j, y_j), $j = t - \tilde{m} + 1, \ldots t$, where $\tilde{m} = \min\{t, M\}$.

Output: new matrix H_t

```
1: Set $H = (s_t^T y_t)/(y_t^T y_t) I$, where $s_t$ and $y_t$ are computed in step 15 of Algorithm 1.
2: for  $j = t - \tilde{m} + 1, \ldots, t$ do
3:     $\rho_j = 1/y_j^T s_j$.
4:     Apply BFGS formula:

   (2.5)          $H \leftarrow (I - \rho_j s_j y_j^T) H (I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T$

5: end for
6: return $H_t \leftarrow H$
```

In practice, the quasi-Newton matrix H_t is not formed explicitly; to compute the product $H_t \nabla F_{\mathcal{S}}(w^k)$ in step 10 of Algorithm 1 one employs a formula based on the structure of the 2-rank BFGS update. This formula, commonly called the two-loop recursion, computes the step directly from the correction pairs and stochastic gradient as described in [18, section 7.2].

In summary, the algorithm builds upon the strengths of BFGS updating, but deviates from the classical method in that the correction pairs (s, y) are based on *subsampled* Hessian-vector products computed at regularly spaced intervals, which amortize their cost. Our task in the remainder of the paper is to argue that even with the extra computational cost of Hessian-vector products (2.2) and the extra cost of computing the iteration (1.6), the SQN method is competitive with the SGD method in terms of computing time (even in the early stages of the optimization), and is able to find a lower objective value.

2.1. Computational cost. Let us compare the cost of the SGD method

$$w^{k+1} = w^k - \frac{\beta}{k}\nabla F_{\mathcal{S}}(w^k) \qquad \text{(SGD)} \tag{2.6}$$

and the SQN method

$$w^{k+1} = w^k - \frac{\beta}{k}H_t\nabla F_{\mathcal{S}}(w^k) \quad \text{(SQN)} \tag{2.7}$$

given by Algorithm 1.

The quasi-Newton matrix-vector product in (2.7) requires approximately $4Mn$ operations [18]. To measure the cost of the gradient and Hessian-vector computations, let us consider one particular but representative example, namely, the binary classification test problem tested in section 4; see (4.1). In this case, the component function f in (1.2) is given by

$$f(w; x_i, z_i) = z_i \log(c(w; x_i)) + (1 - z_i) \log\left(1 - c(w; x_i)\right),$$

where

$$c(w; x_i) = \frac{1}{1 + \exp(-x_i^T w)}, \quad x_i \in \mathbb{R}^n,\ w \in \mathbb{R}^n,\ z_i \in \{0, 1\}. \tag{2.8}$$

The gradient and Hessian-vector product of f are given by

$$\nabla f(w; x_i, z_i) = (c(w; x_i) - z_i)x_i, \tag{2.9}$$

$$\nabla^2 f(w; x_i, z_i)s = c(w; x_i)(1 - c(w; x_i))(x_i^T s)x_i. \tag{2.10}$$

The evaluation of the function $c(w; x_i)$ requires approximately n operations (where we follow the convention of counting a multiplication and an addition as an operation). Therefore, by (2.9) the cost of evaluating one batch gradient is approximately $2bn$, and the cost of computing the Hessian-vector product $\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)s_t$ is about $3b_H n$. This assumes these two vectors are computed independently. If the Hessian is computed at the same point where we compute a gradient and $b \geq b_H$ then $c(w; x_i)$ can be reused for a savings of $b_H n$.

Therefore, for binary logistic problems the total number of floating point operations of the SQN iteration (2.7) is approximately

$$2bn + 4Mn + 3b_H n/L. \tag{2.11}$$

On the other hand, the cost associated with the computation of the SGD step is only

bn. At first glance it may appear that the SQN method is prohibitively expensive, but this is not the case when using the values for b, b_H, L, and M suggested in this paper. To see this, note that

$$\frac{\text{cost of SQN iteration}}{\text{cost of SGD iteration}} = 1 + \frac{2M}{b} + \frac{2b_H}{3bL}. \tag{2.12}$$

In the experiments reported below, we use $M = 5$, $b = 50, 100, \ldots$, $L = 10$ or 20, and choose $b_H \geq 300$. For such parameter settings, the additional cost of the SQN iteration is small relative to the cost of the SGD method.

For the multiclass logistic regression problem described in section 4.3, the costs of gradient and Hessian-vector products are slightly different. Nevertheless, the relative cost of the SQN and SGD iterations is similar to that given in (2.12).

We should mention in passing that the SQN method can take advantage of parallelism. The Hessian-vector products can be computed in parallel with the main iteration (2.7) if we allow freedom in the choice of the point $\bar{w}_t$ where (2.2) is computed. The choice of this point is not delicate since it suffices to estimate the average curvature of the problem around the current iterate, and hence the computation of (2.2) can lag behind the main iteration. In such a parallel setting, the computational overhead of Hessian-vector products may be negligible.

The SQN method contains several parameters, and we provide the following guidelines on how to select them. First, the minibatch size b is often dictated by the experimental set-up or the computing environment, and we view it as an exogenous parameter. A key requirement in the design of our algorithm is that it should work well with any value of b. Given b, our experiments show that it is most efficient if the per-iteration cost of updating, namely, b_H/L, is less than the cost of the stochastic gradient b, with the ratio Lb/b_H in the range [2,20]. The choice of the parameter M in L-BFGS updating is similar to that in deterministic optimization; the best value is problem dependent but values in the range [4,20] are commonly used.

3. Convergence analysis.

In this section, we analyze the convergence properties of the SQN method. We assume that the objective function F is strongly convex and twice continuously differentiable. The first assumption may appear to be unduly restrictive because in certain settings (such as logistic regression) the component functions $f(w; x_i, z_i)$ in (1.3) are convex, but not strongly convex. However, since the lack of strong convexity can lead to very slow convergence, it is common in practice to either add an ℓ_2 regularization term, or choose the initial point (or employ some other mechanism) to ensure that the iterates remain in a region where the F is strongly convex. If regularization is used, the objective function takes the form

$$\tfrac{1}{2}\sigma\|w\|^2 + \tfrac{1}{N}\sum_{i=1}^{N} f(w; x_i, z_i) \qquad \text{with } \ \sigma > 0, \tag{3.1}$$

and the sampled Hessian (2.3) is

$$\sigma I + \frac{1}{b_H}\sum_{i \in \mathcal{S}_H} \nabla^2 f(w; x_i, z_i). \tag{3.2}$$

In this paper, we do not specify the precise mechanism by which the strong convexity is ensured. The assumptions made in our analysis are as follows.

Assumption 1. (1) The objective function F is twice continuously differentiable.
(2) There exist positive constants λ and Λ such that

$$\lambda I \prec \nabla^2 F_{\mathcal{S}_H}(w) \prec \Lambda I \tag{3.3}$$

for all $w \in \mathbb{R}^n$ and all $\mathcal{S}_H \subseteq \{1, \ldots, N\}$. This implies that the true objective F satisfies

$$\lambda I \prec \nabla^2 F(w) \prec \Lambda I \qquad \forall w \in \mathbb{R}^n. \tag{3.4}$$

(3) There is a constant γ such that, for all $w \in \mathbb{R}^n$,

$$E_\xi[\|\nabla f(w^k, \xi)\|^2] \le \gamma^2. \tag{3.5}$$

These assumptions imply that F has a unique minimizer w^*. If the matrices $\nabla^2 f(w;, x_i, z_i)$ are nonnegative definite and uniformly bounded and we implement ℓ_2 regularization as in (3.1)–(3.2) then part (2) of Assumption 1 is satisfied.

We first show that the Hessian approximations generated by the SQN method have eigenvalues that are uniformly bounded above and away from zero. We note that the same result was shown independently by Mokhtari and Ribeiro in [13].

LEMMA 3.1. *If Assumption* 1 *holds, there exist constants* $0 < \mu_1 \le \mu_2$ *such that the Hessian approximations* $\{H_t\}$ *generated by Algorithm* 1 *satisfy*

$$\mu_1 I \prec H_t \prec \mu_2 I \qquad \textit{for} \;\; t = 1, 2, \ldots. \tag{3.6}$$

Proof. Instead of analyzing the inverse Hessian approximation H_k, we will study the direct Hessian approximation B_k (see (1.5)) because this allows us to easily quote a result from the literature. In this case, the limited memory quasi-Newton updating formula is given as follows:

(i) Set $B_t^{(0)} = \frac{y_t^T y_t}{s_t^T y_t} I$ and $\tilde{m} = \min\{t, M\}$.

(ii) For $i = 0, \ldots, \tilde{m} - 1$ set $j = t - \tilde{m} + 1 + i$ and compute

$$B_t^{(i+1)} = B_t^{(i)} - \frac{B_t^{(i)} s_j s_j^T B_t^{(i)}}{s_j^T B_t^{(i)} s_j} + \frac{y_j y_j^T}{y_j^T s_j}. \tag{3.7}$$

(iii) Set $B_{t+1} = B_t^{(\tilde{m})}$.

By (2.2)

$$s_j = \bar{w}_j - \bar{w}_{j-1}, \qquad y_j = \nabla^2 F_{\mathcal{S}_H}(\bar{w}_j) s_j, \tag{3.8}$$

and thus by (3.3)

$$\lambda \|s_j\|^2 \le y_j^T s_j \le \Lambda \|s_j\|^2. \tag{3.9}$$

Now,

$$\frac{\|y_j\|^2}{y_j^T s_j} = \frac{s_j^T [\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)]^2 s_j}{s_j^T \nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)^2 s_j},$$

and since $\nabla^2 F_{\mathcal{S}_H}(\bar{w}_t)$ is symmetric and positive definite, it has a square root so that

$$\lambda \le \frac{\|y_j\|^2}{y_j^T s_j} \le \Lambda. \tag{3.10}$$

This proves that the eigenvalues of the matrices $B_t^{(0)} = \frac{y_t^T y_t}{s_t^T y_t} I$ at the start of the L-BFGS update cycles are bounded above and away from zero for all t.

Let $\text{Tr}(\cdot)$ denote the trace of a matrix. Then from (3.7), (3.10), and the boundedness of $\{\|B_t^{(0)}\|\}$, and setting $j_i = t - \tilde{m} + i$,

$$\begin{aligned} \text{Tr}(B_{t+1}) &\leq \text{Tr}(B_t^{(0)}) + \sum_{i=1}^{\tilde{m}} \frac{\|y_{j_i}\|^2}{y_{j_i}^T s_{j_i}} \\ &\leq \text{Tr}(B_t^{(0)}) + \tilde{m}\Lambda \\ &\leq M_3 \end{aligned} \tag{3.11}$$

for some positive constant M_3. This implies that the largest eigenvalue of all matrices B_t is bounded uniformly.

We now derive an expression for the determinant of B_t. It is shown by Powell [21] that

$$\begin{aligned} \det(B_{t+1}) &= \det(B_t^{(0)}) \prod_{i=1}^{\tilde{m}} \frac{y_{j_i}^T s_{j_i}}{s_{j_i}^T B_t^{(i-1)} s_{j_i}} \\ &= \det(B_t^{(0)}) \prod_{i=1}^{\tilde{m}} \frac{y_{j_i}^T s_{j_i}}{s_{j_i}^T s_{j_i}} \frac{s_{j_i}^T s_{j_i}}{s_{j_i}^T B_t^{(i-1)} s_{j_i}}. \end{aligned} \tag{3.12}$$

Since by (3.11) the largest eigenvalue of $B_t^{(i)}$ is less than M_3, we have, using (3.9) and the fact that the smallest eigenvalue of $B_t^{(0)}$ is bounded away from zero,

$$\begin{aligned} \det(B_{t+1}) &\geq \det(B_t^{(0)}) \left(\frac{\lambda}{M_3}\right)^{\tilde{m}} \\ &\geq M_4 \end{aligned} \tag{3.13}$$

for some positive constant M_4. This shows that the smallest eigenvalue of the matrices B_t is bounded away from zero, uniformly. Therefore, condition (3.6) is satisfied. □

Our next task is to establish global convergence. Rather than proving this result just for our SQN method, we analyze a more general iteration that covers it as a special case. We do so because the more general result is of interest beyond this paper and we are unaware of a self-contained proof of this result in the literature (cf. [26]).

We consider the Newton-like iteration

$$w^{k+1} = w^k - \alpha^k H_k \nabla f(w^k, \xi^k) \tag{3.14}$$

when applied to a strongly convex objective function $F(w)$. (As in (1.2), we used the notation $\xi = (x, z)$.) We assume that the eigenvalues of $\{H_k\}$ are uniformly bounded above and away from zero, and that $E_{\xi^k}[\nabla f(w^k, \xi^k)] = \nabla F(w^k)$. Clearly Algorithm 1 is a special case of (3.14) in which H_k is constant for L iterations.

For convenience, we define $\alpha^k = \beta/k$ for an appropriate choice of β, rather than assuming well-known and more general conditions $\sum \alpha^k = \infty$, $\sum(\alpha^k)^2 < \infty$. This allows us to provide a short proof similar to the analysis of Nemirovski et al. [17].

THEOREM 3.2. *Suppose that Assumption* 1 *holds. Let* w^k *be the iterates generated by the Newton-like method* (3.14)*, where for* $k = 1, 2, \ldots,$

$$\mu_1 I \prec H_k \prec \mu_2 I, \qquad 0 < \mu_1 \leq \mu_2, \tag{3.15}$$

and

$$\alpha^k = \beta/k \quad \textit{with} \quad \beta > 1/(2\mu_1\lambda).$$

Then for all $k \geq 1$*,*

$$E[F(w^k) - F(w^*)] \leq Q(\beta)/k, \tag{3.16}$$

where

$$Q(\beta) = \max\left\{\frac{\Lambda\mu_2^2\beta^2\gamma^2}{2(2\mu_1\lambda\beta - 1)}, F(w^1) - F(w^*)\right\}. \tag{3.17}$$

Proof. We have that

$$\begin{aligned} F(w^{k+1}) &= F(w^k - \alpha^k H_k \nabla f(w^k, \xi^k)) \\ &\leq F(w^k) + \nabla F(w^k)^T(-\alpha^k H_k \nabla f(w^k, \xi^k)) + \tfrac{\Lambda}{2}\|\alpha^k H_k \nabla f(w^k, \xi^k)\|^2 \\ &\leq F(w^k) - \alpha^k \nabla F(w^k)^T(H_k \nabla f(w^k, \xi^k)) + \tfrac{\Lambda}{2}(\alpha^k \mu_2 \|\nabla f(w^k, \xi^k)\|)^2. \end{aligned}$$

Taking the expectation over all possible values of ξ^k and recalling (3.5) gives

$$\begin{aligned} E_{\xi^k}[F(w^{k+1})] &\leq F(w^k) - \alpha^k \nabla F(w^k)^T H_k \nabla F(w^k) + \tfrac{\Lambda}{2}(\alpha^k\mu_2)^2 E_{\xi^k}[\|\nabla f(w^k, \xi^k)\|]^2 \\ &\leq F(w^k) - \alpha^k \mu_1 \|\nabla F(w^k)\|^2 + \tfrac{\Lambda}{2}(\alpha^k\mu_2)^2\gamma^2. \end{aligned} \tag{3.18}$$

Now, to relate $F(w^k) - F(w^*)$ and $\|\nabla F(w^k)\|^2$, we use the lower bound in (3.4) to construct a minorizing quadratic for F at w^k. For any vector $v \in \mathbb{R}^n$, we have

$$\begin{aligned} F(v) &\geq F(w^k) + \nabla F(w^k)^T(v - w^k) + \tfrac{\lambda}{2}||v - w^k||^2 \\ &\geq F(w^k) + \nabla F(w^k)^T(-\tfrac{1}{\lambda}\nabla F(w^k)) + \tfrac{\lambda}{2}||\tfrac{1}{\lambda}\nabla F(w^k)||^2 \\ &\geq F(w^k) - \tfrac{1}{2\lambda}||\nabla F(w^k)||^2, \end{aligned} \tag{3.19}$$

where the second inequality follows from the fact that $\hat{v} = w^k - \frac{1}{\lambda}\nabla F(w^k)$ minimizes the quadratic $q_k(v) = F(w^k) + \nabla F(w^k)^T(v - w^k) + \frac{\lambda}{2}||v - w^k||^2$. Setting $v = w^*$ in (3.19) yields

$$2\lambda[F(w^k) - F(w^*)] \leq \|\nabla F(w^k)\|^2,$$

which together with (3.18) yields

$$E_{\xi^k}[F(w^{k+1}) - F(w^*)] \leq F(w^k) - F(w^*) - 2\alpha^k\mu_1\lambda[F(w^k - F(w^*)] + \tfrac{\Lambda}{2}(\alpha^k\mu_2)^2\gamma^2. \tag{3.20}$$

Let us define ϕ_k to be the expectation of $F(w^k) - F(w^*)$ over all choices $\{\xi^1, \xi^2, \ldots, \xi^{k-1}\}$ starting at w^1, which we write as

$$\phi_k = E[F(w^k) - F(w^*)]. \tag{3.21}$$

Then (3.20) yields

$$\phi_{k+1} \leq (1 - 2\alpha^k\mu_1\lambda)\phi_k + \tfrac{\Lambda}{2}(\alpha^k\mu_2)^2\gamma^2. \tag{3.22}$$ □

We prove the desired result (3.16) by induction. The result clearly holds for $k = 1$. Assuming it holds for some value of k, inequality (3.22), definition (3.17), and the choice of α^k imply

$$\begin{aligned}\phi_{k+1} &\leq \left(1 - \frac{2\beta\mu_1\lambda}{k}\right)\frac{Q(\beta)}{k} + \frac{\Lambda\mu_2^2\beta^2\gamma^2}{2k^2}\\ &= \frac{(k-2\beta\mu_1\lambda)Q(\beta)}{k^2} + \frac{\Lambda\mu_2^2\beta^2\gamma^2}{2k^2}\\ &= \frac{(k-1)Q(\beta)}{k^2} - \frac{2\beta\mu_1\lambda - 1}{k^2}Q(\beta) + \frac{\Lambda\mu_2^2\beta^2\gamma^2}{2k^2}\\ &\leq \frac{Q(\beta)}{k+1}.\end{aligned}$$

We can now establish the main result of this section.

Corollary 3.3. *Suppose that Assumption* 1 *holds. Let* $\{w^k\}$ *be the iterates generated by Algorithm* 1. *Then there is a constant* μ_1 *such that* $\|H_k^{-1}\| \leq 1/\mu_1$ *, and if the step length is chosen by*

$$\alpha^k = \beta/k \quad \textit{where} \quad \beta > 1/(2\mu_1\lambda) \quad \forall k,$$

it follows that

$$E[F(w^k) - F(w^*)] \leq Q(\beta)/k \tag{3.23}$$

for all k, where

$$Q(\beta) = \max\left\{\frac{\Lambda\mu_2^2\beta^2\gamma^2}{2(2\mu_1\lambda\beta - 1)}, F(w^1) - F(w^*)\right\}. \tag{3.24}$$

Proof. Lemma 3.1 ensures that the Hessian approximation satisfies (3.6). Now, the iteration in step 10 of Algorithm 1 is a special case of iteration (3.14). Therefore, the result follows from Theorem 3.2. ▯

As in the classical L-BFGS method for deterministic optimization where one cannot show that the rate of convergence is faster than the steepest descent method, our algorithm does not improve upon the convergence rate of the stochastic gradient method. Nevertheless, as we show experimentally in the next section, the quasi-Newton approach yields a significant reduction in both iterations and overall computational cost.

4. Numerical experiments.

In this section, we compare the performance of the SGD method (2.6) and the SQN method (2.7) on three test problems of the form (1.2)–(1.3) arising in supervised machine learning. The parameter $\beta > 0$ is fixed at the beginning of each run, as discussed below, and the SQN method is implemented as described in Algorithm 1. The comparisons will be based on sample complexity, i.e., on the amount of evaluation of each sampled function and its derivatives; see point 7 below.

It is well known amongst the optimization and machine learning communities that the SGD method can be improved by choosing the parameter β via a set of problem dependent heuristics [20, 28]. In some cases, β_k (rather than β) is made to vary during the course of the iteration, and could even be chosen so that β_k/k is constant, in which case only convergence to a neighborhood of the solution is guaranteed [16]. There is, however, no generally accepted rule for choosing β_k, so our testing approach is to consider the simple strategy of selecting the (constant) β so as to give a good performance for each problem.

Specifically, in the experiments reported below, we tried several values for β in (2.6) and (2.7) and chose a value for which increasing or decreasing it by a fixed increment results in inferior performance. This allows us to observe the effect of the quasi-Newton Hessian approximation H_k in a controlled setting, without the clutter introduced by elaborate step-length strategies for β_k.

In the figures provided below, we use the following notation.

1. n: the number of variables in the optimization problem, i.e., $w \in \mathbb{R}^n$.
2. N: the number of training points in the dataset.
3. b: size of the batch used in the computation of the stochastic gradient $\nabla F_{\mathcal{S}}(w^k)$ defined in (1.4), i.e., $b = |\mathcal{S}|$.
4. b_H: size of the batch used in the computation of Hessian-vector products (2.2) and (2.3), i.e., $b_H = |\mathcal{S}_H|$.
5. L: controls the frequency of limited memory BFGS updating. Every L iterations a new curvature pair (2.2) is formed and the oldest pair is removed.
6. M: memory used in limited memory BFGS updating.
7. adp: accessed data points. At each iteration the SGD method evaluates the stochastic gradient $\nabla F_{\mathcal{S}}(w^k)$ using b randomly chosen training points (x_i, z_i), so we say that the iteration accessed b data points. On the other hand, an iteration of the stochastic BFGS method accesses $b + b_H/L$ data points.
8. iteration: in some graphs we compare SGD and SQN iteration by iteration (in addition to comparing them in terms of adps).
9. epoch: one complete pass through the dataset.

In our experiments, the stochastic gradient (1.4) is formed by randomly choosing b training points from the dataset without replacement. This process is repeated every epoch, which guarantees that all training points are equally used when forming the stochastic gradient. Independently of the stochastic gradients, the Hessian-vector products are formed by randomly choosing b_H training points from the dataset without replacement.

4.1. Experiments with synthetic datasets. We first test our algorithm on a binary classification problem. The objective function is given by

$$F(w) = -\frac{1}{N}\sum_{i=1}^{N} z_i \log(c(w;x_i)) + (1-z_i)\log\left(1 - c(w;x_i)\right), \tag{4.1}$$

where $c(w;x_i)$ is defined in (2.8).

The training points were generated randomly as described in [14] with $N = 7000$ and $n = 50$. To establish a reference benchmark with a well-known algorithm, we used the particular implementation [14] of one of the coordinate descent (CD) methods of Tseng and Yun [27].

Figure 1 reports the performance of SGD (with $\beta = 7$) and SQN (with $\beta = 2$), as measured by adps. Both methods use a gradient batch size of $b = 50$; for SQN we display results for two values of the Hessian batch size b_H, and set $M = 10$ and $L = 10$. The vertical axis, labeled `fx`, measures the value of the objective (4.1); the dotted black line marks the best function value obtained by the CD method mentioned above. We observe that the SQN method with $b_H = 300$ and 600 outperforms SGD, and obtains the same or better objective value than the CD method.

In Figure 2 we explore the effect of the memory size M. Increasing M beyond 1 and 2 steadily improves the performance of the SQN algorithm, both during the first few epochs (left figure), and after letting the algorithm run for many epochs (right figure). For this problem, a large memory size is helpful in the later stages of the run.

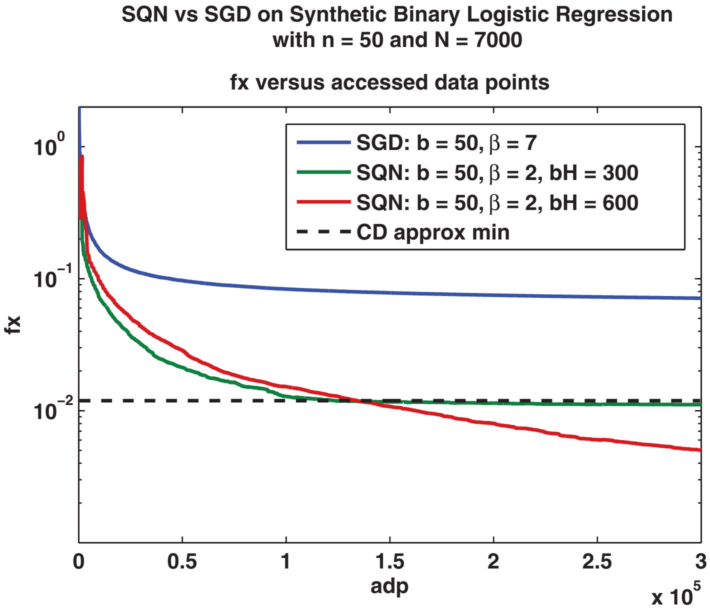


FIG. 1. *Illustration of SQN and SGD on the synthetic dataset. The dotted black line marks the best function value obtained by the CD method. For SQN we set $M = 10$, $L = 10$, and $b_H = 300$ or* 600.

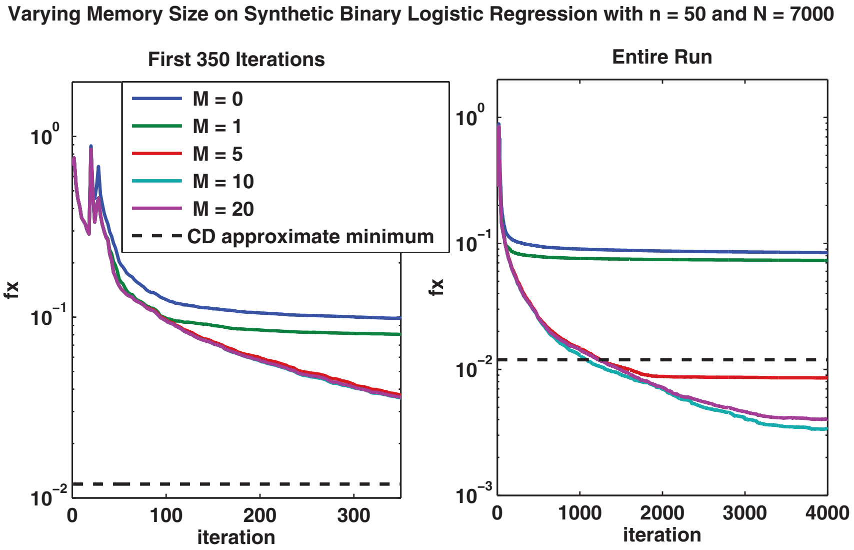


FIG. 2. *Effect of the memory size M in the SQN method. The figure on the left reports the first 4 epochs, while the figure on the right lets the algorithm run for more than 70 epochs to observe if the beneficial effect of increasing M is sustained. Parameters settings are $b = 50$, $b_H = 600$, and* $L = 10$.

4.2. RCV1 data set. The RCV1 dataset [11] is a composition of newswire articles produced by Reuters from 1996–1997. Each article was manually labeled into 4 different classes: Corporate/Industrial, Economics, Government/Social, and Markets. For the purpose of classification, each article was then converted into a

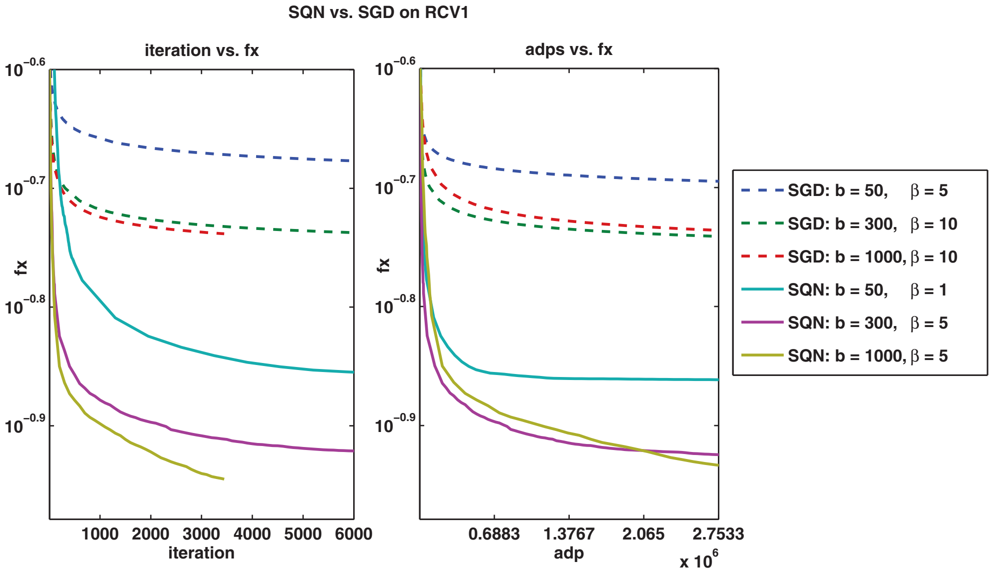


FIG. 3. *Illustration on RCV1 problem. For SGD and SQN, b is set to either* 50, 300, *or* 1000, *and for SQN we use $b_H = 1000$, $M = 5$, and $L = 20$. The figures report training error as a function of iteration count or adps. In the rightmost graph the tick marks on the x-axis (at* 0.6882, 1.3767, ...*) denote the epochs of SGD.*

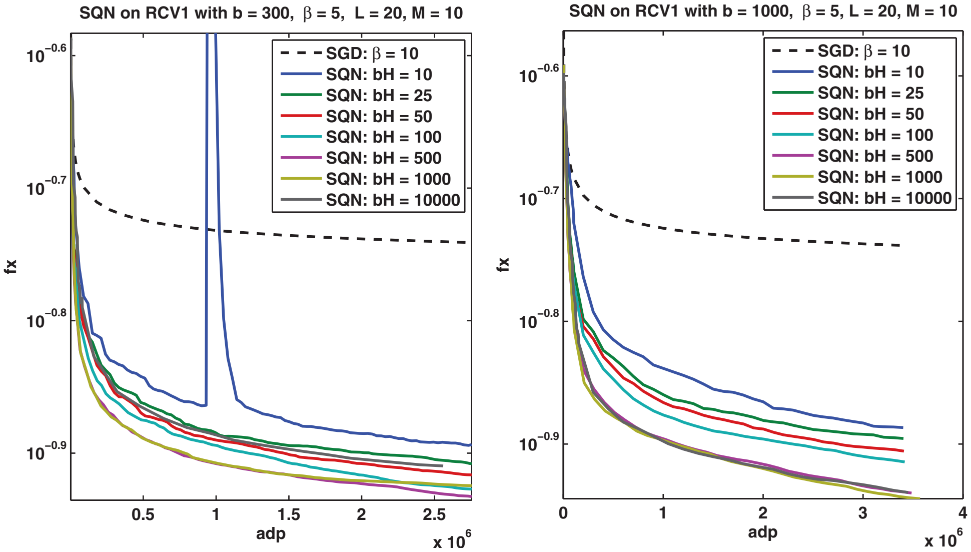
boolean feature vector with a 1 representing the appearance of a given word. Post word stemming, this gives each feature vector a dimension of $n = 112919$.

Each data point $x_i \in [0,1]^n$ is extremely sparse, with an average of 91 (.013%) nonzero elements. There are $N = 688329$ training points. We consider the binary classification problem of predicting whether or not an article is in the fourth class, Markets, and accordingly we have labels $z_i \in \{0,1\}$. We use logistic regression to model the problem, and define the objective function by (4.1).

In our numerical experiments with this problem, we used gradient batch sizes of $b = 50$, 300, or 1000, which, respectively, comprise .0073%, .044%, and .145% of the dataset. The frequency of quasi-Newton updates was set to $L = 20$, a value that balances the aims of quickly retrieving curvature information and minimizing computational costs. For the SGD method we chose $\beta = 5$ when $b = 50$, and $\beta = 10$ when $b = 300$ or 1000; for the SQN method (2.7) we chose $\beta = 1$ when $b = 50$, and $\beta = 5$ when $b = 300$ or 1000, and we set $b_H = 1000$.

Figure 3 reports the performance of the two methods as measured by either iteration count or adps. As before, the vertical axis, labeled `fx`, measures the value of the objective (4.1). Figure 3 shows that for each batch size, the SQN method outperforms SGD, and both methods improve as batch size increases. We observe that using $b = 300$ or 1000 yields different relative outcomes for the SQN method when measured in terms of iterations or adp: a batch size of 300 provides the fastest initial decrease in the objective, but that method is eventually overtaken by the variant with the larger batch size of 1000.

Figure 4 illustrates the effect of varying the Hessian batch size b_H from 10 to 10000, while keeping the gradient batch size b fixed at 300 or 1000. For $b = 300$ (Figure 4(a)) increasing b_H improves the performance of SQN, in terms of adp, up until $b_H = 1000$, where the benefits of the additional accuracy in the Hessian approximation



(a) Varying b_H for $b = 300$

(b) Varying b_H for $b = 1000$

FIG. 4. *Varying Hessian batch size parameter* b_H *on the RCV*1 *dataset for gradient batch values* b *of* 300 *and* 1000*. All other parameters in the SQN method are held constant at* $L = 20$*,* $M = 10$*, and* $\beta = 5$*.*

do not outweigh the additional computational cost. In contrast, Figure 4(b) shows that for $b = 1000$, a high value for b_H, such as 10000, can be effectively used since the cost of the Hessian-vector product relative to the gradient is lower. One of the conclusions drawn from this experiment is that there is much freedom in the choice of b_H, and that only a small subsample of the data (e.g., $b_H = 100$) is needed for the SQN approach to yield benefits.

One should guard, however, against the use of very small values for b_H, as seen in the large blue spike in Figure 4(a) corresponding to $b_H = 10$. To understand this behavior, we monitored the angle between the vectors s and y and observed that it approached 90° between iteration 3100 and 3200, which is where the spike occurred. Since the term $s^T y$ enters in the denominator of the BFGS update formula (2.5), this led to a very large and poor step. Monitoring $s^T y$ (relative to, say, $s^T B s$) can be a useful indicator of a potentially harmful update; one can increase b_H or skip the update when this number is smaller than a given threshold.

The impact of the memory size parameter M is shown in Figure 5. The results improve consistently as M increases, but beyond $M = 2$ these improvements are rather small, especially in comparison to the results in Figure 2 for the synthetic data. The reasons for this difference are not clear, but for the deterministic L-BFGS method the effect of M on performance is known to be problem dependent. We observe that performance with a value of $M = 0$, which results in a Hessian approximation of the form $H_t = \frac{s_t^T y_t}{y_t^T y_t} I$, is poor and also unstable in early iterations, as shown by the spikes in Figure 5.

To gain a better understanding of the behavior of the SQN method, we also monitored the following two errors:

$$\text{GradError} = \frac{\|\nabla F(w) - \nabla F_{\mathcal{S}}(w)\|_2}{\|\nabla F(w)\|_2} \tag{4.2}$$

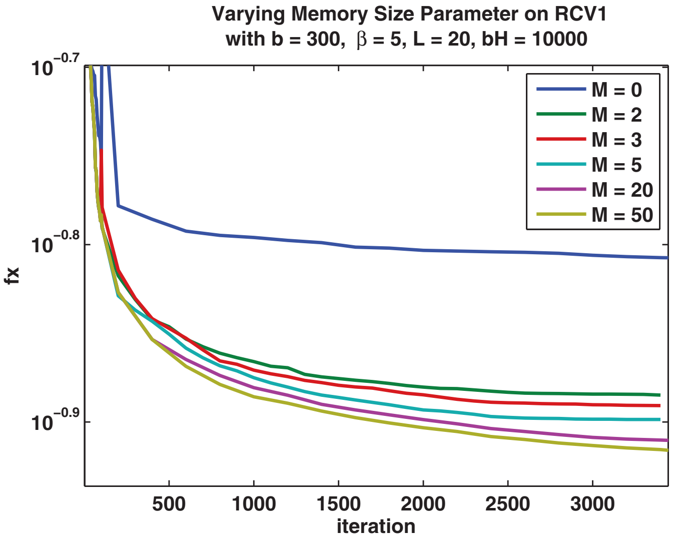


FIG. 5. *Impact of the memory size parameter on the RCV1 dataset. M is varied between 0 and 50 while all other parameters are held constant at $b = 300$, $L = 20$, and $b_H = 10000$.*

and

$$\text{HvError} = \frac{\left\|\nabla^2 F(\bar{w}_I)(\bar{w}_I - \bar{w}_J) - \nabla^2 F_{\mathcal{S}_H}(\bar{w}_I)(\bar{w}_I - \bar{w}_J)\right\|_2}{\left\|\nabla^2 F(\bar{w}_I)(\bar{w}_I - \bar{w}_J)\right\|_2}. \tag{4.3}$$

The quantities $\nabla F(w)$ and $\nabla^2 F(\bar{w}_I)(\bar{w}_I - \bar{w}_J)$ are computed with the entire data set, as indicated by (4.1). Therefore, the ratios above report the relative error in the stochastic gradient used in (2.7) and the relative error in the computation of the Hessian-vector product (2.2).

Figure 6 displays these relative errors for various batch sizes b and b_H, along with the norms of the stochastic gradients. These errors were calculated every 20 iterations during a *single run* of SQN with the following parameter settings: $b = 300$, $L = 20$, $M = 5$, and $b_H = 688329$. Batch sizes larger than $b = 10000$ exhibit nonstochastic behavior in the sense that all relative errors are less than one, and the norm of these approximate gradients decreases during the course of the iteration. Gradients with a batch size less than 10000 have relative errors greater than 1, and their norm does not exhibit decrease over the course of the run.

The leftmost figure also shows that the ℓ_2 norms of the stochastic gradients decrease as the batch size b increases, i.e., there is a tendency for inaccurate gradients to have a larger norm, as expected from the geometry of the error.

Figure 6 indicates that the Hessian-vector errors stay relatively constant throughout the run and have smaller relative error than the gradient. As discussed above, some accuracy here is important while it is not needed for the batch gradient.

4.3. A speech recognition problem.

The speech dataset, provided by Google, is a collection of feature vectors representing 10 millisecond frames of speech with a corresponding label representing the phonetic state assigned to that frame. Each feature x_i has a dimension of $NF = 235$ and has corresponding label $z_i \in C = \{1, 2, \dots, 129\}$. There are a total of $N = 191607$ samples; the number of variables is $n = NF \times |C| = 30315$.

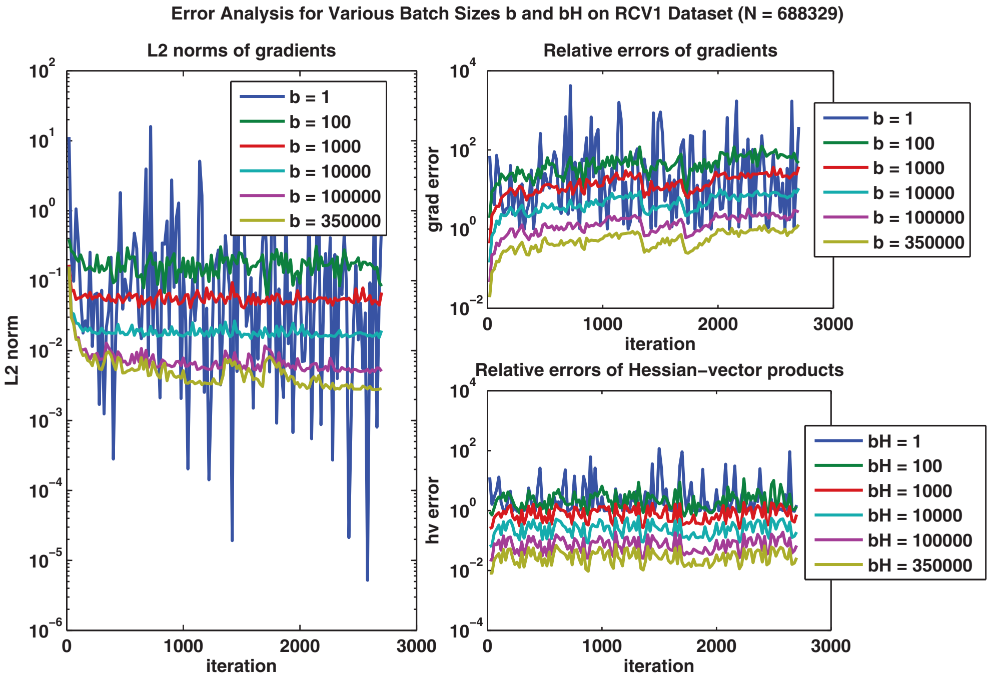


FIG. 6. *Error plots for RCV1 dataset. The figure on the left plots* $\|\nabla F(w)\|_2$ *for various values of* b*. The figures on the right display the errors* (4.2) *and* (4.3)*. The errors were calculated every* 20 *iterations during a single run of SQN with parameters* $b = 300$*,* $L = 20$*,* $M = 5$*, and* $b_H = 688329$*.*

The problem is modeled using multiclass logistic regression. The unknown parameters are assembled in a matrix $W \in \mathbb{R}^{|C|\times NF}$, and the objective is given by

$$F(W) = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\exp(W_{z_i}x_i)}{\sum_{j\in\mathcal{C}}\exp(W_j x_i)}\right), \tag{4.4}$$

where $x_i \in \mathbb{R}^{NF\times 1}$, z_i is the index of the correct class label, and $W_{z_i} \in \mathbb{R}^{1\times NF}$ is the row vector corresponding to the weights associated with class z_i.

Figure 7 displays the performance of SGD and SQN for $b = 100$ and 500 (which represent approximately 0.05%, and 0.25% of the dataset). For the SGD method, we chose the step length $\beta = 10$ for both values of b; for the SQN method we set $\beta = 2$, $L = 10,\ M = 5,\ b_H = 1000$.

We observe from Figure 7 that SQN improves upon SGD in terms of adp, both initially and in finding a lower objective value. Although the number of SQN iterations decreases when b is increased from 100 to 500, in terms of computational cost the two versions of the SQN method yield almost identical performance.

The effect of varying the Hessian batch size b_H is illustrated in Figure 8. The figure on the left shows that increasing b_H improves the performance of SQN, as measured by iterations, but only marginally from $b_H = 1000$ to 10000. Once the additional computation cost of Hessian-vector products is accounted for, we observe from the figure on the right that $b_H = 100$ is as effective as $b_H = 1000$. Once more, we conclude that only a small subset of data points $\mathcal{S}_H$ is needed to obtain useful curvature information in the SQN method.

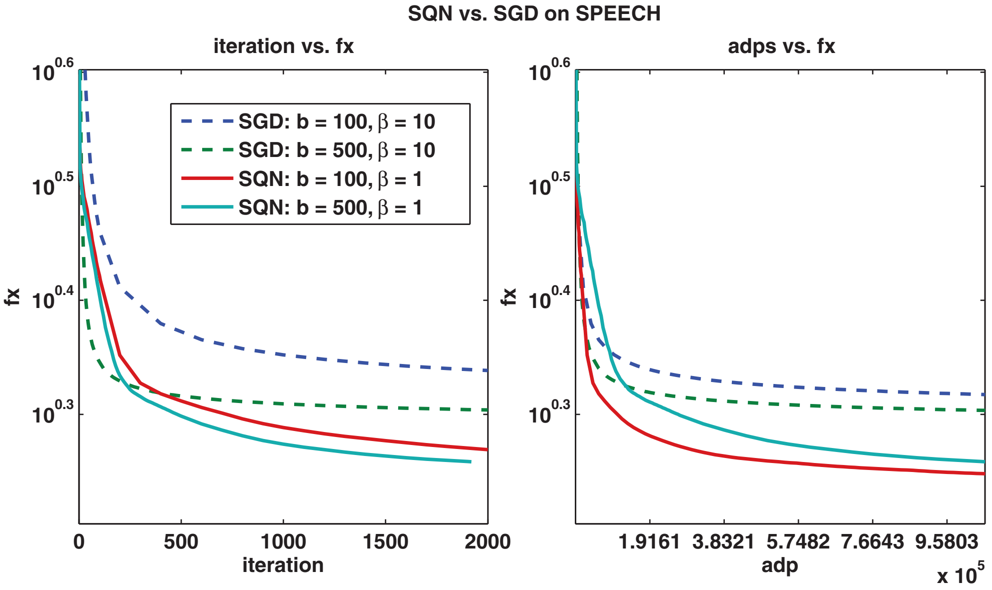


FIG. 7. *Illustration of SQN and SGD on the SPEECH dataset. The gradient batch size b is 100 or 500, and for SQN we use $b_H = 1000$, $M = 5$, and $L = 10$. In the rightmost graph, the tick marks on the x-axis denote the epochs of the SGD method.*

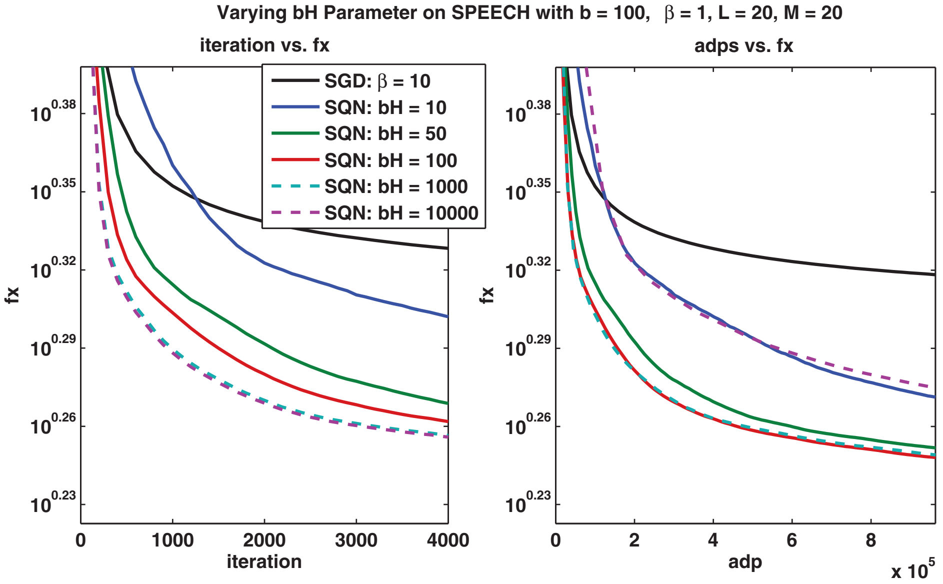


FIG. 8. *Varying Hessian batch size parameter b_H on SPEECH dataset. All other parameters are held constant at $b = 100$, $L = 20$, $M = 20$.*

Figure 9 illustrates the impact of increasing the memory size M from 0 to 20 for the SQN method. A memory size of zero leads to a marked degradation of performance. Increasing M from 0 to 5 improves SQN, but values greater than 5 yield no measurable benefit.

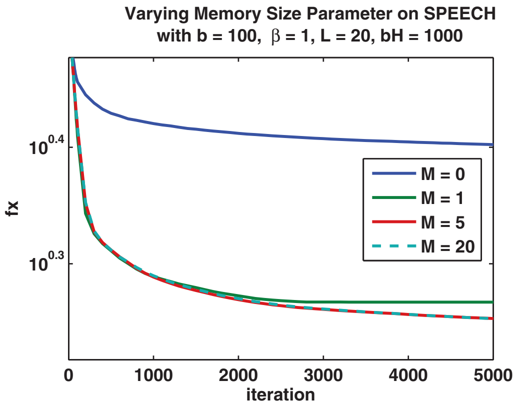
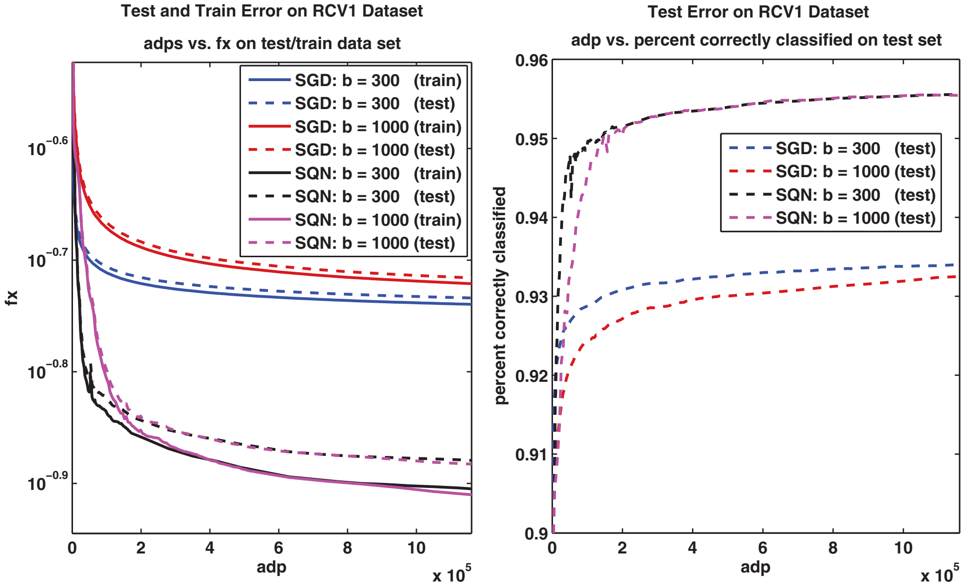


FIG. 9. *Performance on the SPEECH dataset with varying memory size M. All other parameters are held constant at $b = 100$, $L = 20$, $b_H = 1000$.*



(a) Training and test error on the objective

(b) Test error for percent correctly classified

FIG. 10. *Illustration of the generalization error on the RCV1 dataset. For both SGD and SQN, b is set to 300 or 1000; for SQN we set $b_H = 1000$, $M = 5$, and $L = 20$.*

4.4. Generalization error. The primary focus of this paper is on the minimization of training error (1.3), but it is also interesting to explore the performance of the SQN method in terms of generalization (testing) error. For this purpose we consider the RCV1 dataset, and in Figure 10 we report the performance of algorithms SQN and SGD with respect to unseen data (dotted lines). Both algorithms were trained using 75% of the data and then tested on the remaining 25% (the test set). In Figure 10(a), the generalization error is measured in terms of decrease of the objective (4.1) over the test set, and in Figure 10(b), in terms of the percent of correctly classified data points from the test set. The first measure complements the latter in the sense that it takes into account the confidence of the correct predictions and the inaccuracies wrought

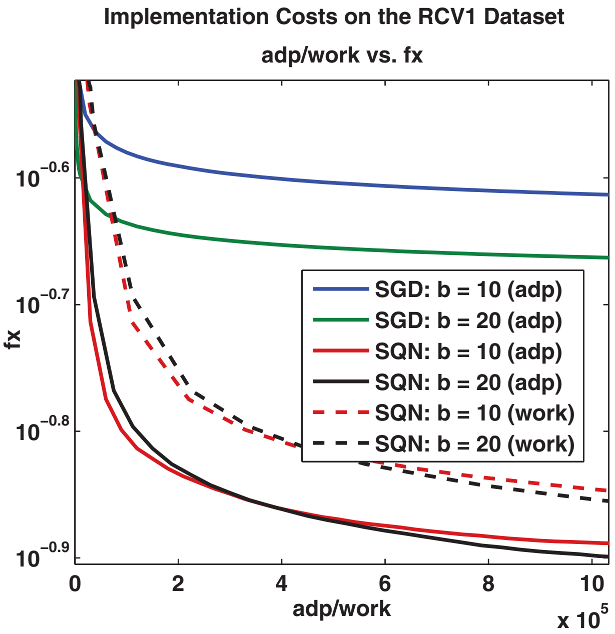


FIG. 11. *Comparison using $b = 10,\ 20$ on RCV1. The solid lines measure performance in terms of adp and the dotted lines measure performance in terms of total computational work* (2.11) *(scaled by a factor of $1/n$). For SQN we set $M = 5$, $b_H = 1000$, $L = 200$, $\beta = 1$, and for SGD we set $\beta = 5$.*

by the misclassifications. Recall that there are 2 classes in our RCV1 experiments, so random guessing would yield a percentage of correct classification of 0.5.

As expected, the objective on the training set is lower than the objective on the test set, but not by much. These graphs suggests that overfitting is not occurring since the objective on the test set decreases monotonically. The performance of the SQN method is clearly very good on this problem.

4.5. Small minibatches. In the experiments reported in the sections 4.2 and 4.3, we used fairly large gradient batch sizes, such as $b = 50, 100, 1000$, because they gave a good performance for both the SGD and SQN methods on our test problems. Since we set $M = 5$, the cost of the multiplication $H_t \nabla F_{\mathcal{S}}(w^k)$ (namely, $4Mn = 20n$) is small compared to the cost of bn for a batch gradient evaluation. We now explore the efficiency of the SQN method for smaller values of the batch size b.

In Figure 11 we report results for the SGD and SQN methods for problem RCV1 for $b = 10$ and 20. We use two measures of performance: total *computational work* and adp. For the SQN method, the work measure is given by (2.11), which includes the evaluation of the gradient (1.4), the computation of the quasi-Newton step (2.7), and the Hessian-vector products (2.2).

In order to compare total work and adp on the same figure, we scale the work by $1/n$. The solid lines in Figure 11 plot the objective value versus adp, while the dotted lines plot function value versus total work. We observe from Figure 11 that, even for small b, the SQN method outperforms SGD by a significant margin in spite of the additional Hessian-vector product cost. Note that in this experiment the $4Mn$ cost of computing the steps is still less than half the total computational cost (2.11) of the SQN iteration.

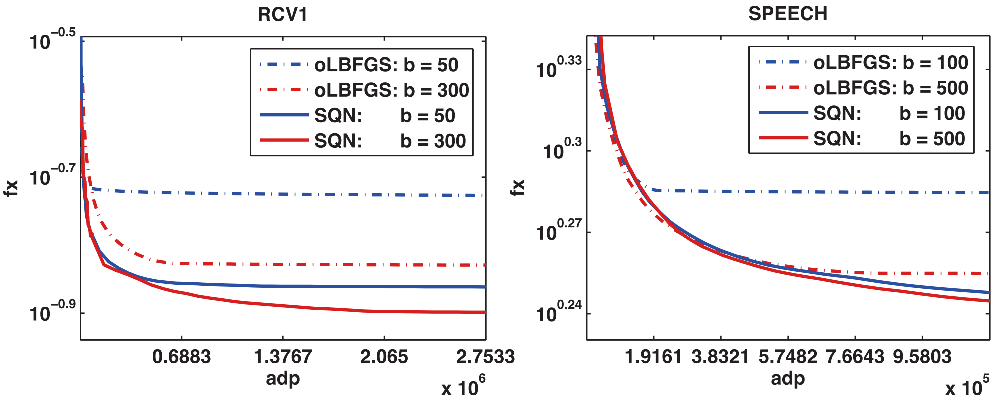


FIG. 12. *Comparison of oLBFGS (dashed lines) and SQN (solid lines) in terms of adps. For the RCV1 dataset gradient batches are set to $b = 50$ or 300 for both methods; additional parameter settings for SQN are $L = 20$, $b_H = 1000$, $M = 10$. For the Speech dataset we set to $b = 100$ or 500; and for SQN we set $L = 10$, $b_H = 1000$, $M = 10$.*

In this experiment, it was crucial to update the quasi-Newton matrix infrequently ($L = 200$), as this allowed us to employ a large value of b_H at an acceptable cost. In general, the parameters L, M, and b_H provide much freedom in adapting the SQN method to a specific application.

4.6. Comparison to the oLBFGS method.

We also compared our algorithm to the oLBFGS method [25], which is the best known stochastic quasi-Newton method in the literature. It is of the form (1.6) but differs from our approach in three crucial respects: the L-BFGS update is performed at every iteration, the curvature estimate is calculated using gradient differencing, and the sample size for gradient differencing is the same as the sample size for the stochastic gradient. This approach requires two gradient evaluations per iteration; it computes

$$w^{k+1} = w^k - \alpha^k H_k \nabla F_{\mathcal{S}_k}(w^k), \quad s_k = w^k - w^{k-1}, \quad y_k = \nabla F_{\mathcal{S}_{k-1}}(w^k) - \nabla F_{\mathcal{S}_{k-1}}(w^{k-1}),$$

where we have used subscripts $\mathcal{S}$ to indicate the sample used in the computation of the gradient. The extra gradient evaluation is similar in cost to our Hessian-vector product, but we compute that product only every L iterations. Thus, the oLBFGS method is analogous to our algorithm with $L = 1$ and $b = b_H$, which as the numerical results below show, is not an efficient allocation of effort. In addition, the oLBFGS method is limited in the choice of samples $\mathcal{S}$ because, when these are small, the Hessian approximations may be of poor quality.

We implemented the oLBFGS method as described in [25], with the following parameter settings: (i) we found it to be unnecessary to add a damping parameter to the computation y_k, and thus set $\lambda = 0$ in the reset $y_k \leftarrow y_k + \lambda s_k$; (ii) the parameter ϵ used to rescale the first iteration, $w^1 = w^0 - \epsilon\alpha^k \nabla F_{\mathcal{S}_0}(w^0)$, was set to $\epsilon = 10^{-6}$; (iii) the initial choice of scaling parameter in Hessian updating (see step 1 of Algorithm 2) was the average of the quotients $s_i^T y_i / y_i^T y_i$ averaged over the last M iterations, as recommended in [25].

Figure 12 compares our SQN method to the aforementioned oLBFGS on our two realistic test problems, in terms of adps. We observe that SQN has overall better performance, which is more pronounced for smaller batch sizes.

5. Related work.

Various stochastic quasi-Newton algorithms have been proposed in the literature [25, 12, 4, 23], but have not been entirely successful. The methods in [25] and [12] use the BFGS framework; the first employs an L-BFGS implementation, as mentioned in the previous section, and the latter uses a regularized BFGS matrix. Both methods enforce uniformity in gradient differencing by resampling data points so that two consecutive gradients are evaluated with the same sample $\mathcal{S}$; this strategy requires an extra gradient evaluation at each iteration. The algorithm presented in [4] uses SGD with a diagonal rescaling matrix based on the secant condition associated with quasi-Newton methods. Similarly to our approach, [4] updates the rescaling matrix at fixed intervals in order to reduce computational costs. A common feature of [25, 12, 4] is that the Hessian approximation might be updated with a high level of noise.

A two-stage online Newton strategy is proposed in [3]. The first stage runs averaged SGD with a step size of order $O(1/\sqrt{k})$, and the second stage minimizes a quadratic model of the objective function using SGD with a constant step size. The second stage effectively takes one Newton step, and employs Hessian-vector products in order to compute stochastic derivatives of the quadratic model. This method is significantly different from our quasi-Newton approach.

A stochastic approximation method that has shown to be effective in practice is AdaGrad [9]. The iteration is of the form (1.5), where B_k is a diagonal matrix that estimates the diagonal of the squared root of the uncentered covariance matrix of the gradients; it is shown in [9] that such a matrix minimizes a regret bound. The algorithm presented in this paper is different in nature from AdaGrad in that it employs a full (nondiagonal) approximation to the Hessian $\nabla^2 F(w)$.

Amari [1] popularized the idea of incorporating information from the geometric space of the inputs into online learning with his presentation of the natural gradient method. This method seeks to find the steepest descent direction in the feature space x by using the Fisher information matrix, and is shown to achieve asymptotically optimal bounds. The method does, however, require knowledge of the underlying distribution of the training points (x, z), and the Fisher information matrix must be inverted. These concerns are addressed in [19], which presents an adaptive method for computing the inverse Fisher information matrix in the context of multilayer neural networks.

The authors of TONGA [24] interpret natural gradient descent as the direction that maximizes the probability of reducing the generalization error. They outline an online implementation using the uncentered covariance matrix of the empirical gradients that is updated in a weighted manner at each iteration. Additionally, they show how to maintain a low rank approximation of the covariance matrix so that the cost of the natural gradient step is $O(n)$. In [23] it is argued that an algorithm should contain information about both the Hessian and covariance matrix, maintaining that that covariance information is needed to cope with the variance due to the space of inputs, and Hessian information is useful to improve the optimization.

Our algorithm may appear at first sight to be similar to the method proposed by Byrd et al. [8, 7], which also employs Hessian-vector products to gain curvature information. We note, however, that the algorithms are different in nature, as the algorithm presented here operates in the stochastic approximation regime, whereas [8, 7] is a batch method.

6. Final remarks.

In this paper, we presented a quasi-Newton method that operates in the stochastic approximation regime. It is designed for the minimization of convex stochastic functions, and was tested on problems arising in machine learning. In contrast to previous attempts at designing stochastic quasi-Newton methods, our approach does not compute gradient differences from consecutive iterations to gather curvature information; instead it computes (subsampled) Hessian-vector products at regular intervals to obtain this information in a stable manner. We have noted that Hessian-vector products can be replaced by the difference (2.4), but at the cost of extra computation.

Our numerical results suggest that the method does more than rescale the gradient, i.e., that its improved performance over the SGD method of Robbins–Monro is the result of incorporating curvature information in the form of a full matrix.

The practical success of the algorithm relies on the fact that the batch size b_H for Hessian-vector products can be chosen large enough to provide useful curvature estimates, while the update spacing L can be chosen large enough (say $L = 20$) to amortize the cost of Hessian-vector products, and make them affordable. Similarly, there is a wide range of values for the gradient batch size b that makes the overall quasi-Newton approach (1.6) viable.

We established global convergence of the algorithm on strongly convex objective functions. Our numerical results indicate that the algorithm is more effective than the best known stochastic quasi-Newton method (oLBFGS [25]) and suggest that it holds much promise for the solution of large-scale problems arising in stochastic optimization. Although we presented and analyzed the algorithm in the convex case, our approach is applicable to nonconvex problems provided it employs a mechanism for ensuring that the condition $s_t^T y_t > 0$ is satisfied.

REFERENCES

[1] S.-I. Amari, *Natural gradient works efficiently in learning*, Neural Comput., 10 (1998), pp. 251–276.

[2] S. Asmussen and P. W. Glynn, *Stochastic Simulation: Algorithms and Analysis*, Stoch. Model. Appl. Probab. 57, Springer, New York, 2007.

[3] F. Bach and E. Moulines, *Non-strongly-convex smooth stochastic approximation with convergence rate $o(1/n)$*, in Advances in Neural Information Processing Systems 26, Curran, Red Hook, NY, 2013, pp. 773–781.

[4] A. Bordes, L. Bottou, and P. Gallinari, *SGD-QN: Careful quasi-Newton stochastic gradient descent*, J. Mach. Learn. Res., 10 (2009), pp. 1737–1754.

[5] L. Bottou and O. Bousquet, *The tradeoffs of large scale learning*, in Advances in Neural Information Processing Systems 20, J. C. Platt, D. Koller, Y. Singer, and S. Roweis, eds., MIT Press, Cambridge, MA, 2008, pp. 161–168.

[6] L. Bottou and Y. LeCun, *Large scale online learning*, in Advances in Neural Information Processing Systems 16, S. Thrun, L. Saul, and B. Schölkopf, eds., MIT Press, Cambridge, MA, 2004.

[7] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, *Sample size selection in optimization methods for machine learning*, Math. Program., 134 (2012), pp. 127–155.

[8] R. H Byrd, G. M Chin, W. Neveitt, and J. Nocedal, *On the use of stochastic Hessian information in optimization methods for machine learning*, SIAM J. Optim., 21 (2011), pp. 977–995.

[9] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res., 12 (2011), pp. 2121–2159.

[10] R. Fletcher, *Practical Methods of Optimization*, 2nd ed., Wiley, Chichester, 1987.

[11] D. D Lewis, Y. Yang, T. G. Rose, and F. Li, *Rcv1: A new benchmark collection for text categorization research*, J. Mach. Learn. Res., 5 (2004), pp. 361–397.

[12] A. Mokhtari and A. Ribeiro, *Regularized stochastic BFGS algorithm*, in IEEE Global Conference on Signal and Information Processing, IEEE, Piscataway, NJ, 2013.

[13] A. Mokhtari and A. Ribeiro, *Global convergence of online limited memory bfgs*, preprint, arXiv:1409.2045, 2014.

[14] I. Mukherjee, K. Canini, R. Frongillo, and Y. Singer, *Parallel boosting with momentum*, in ECML PKDD 2013, Part III, Lecture Notes in Comput. Sci. 8190, Springer, Heidelberg, 2013, pp. 17–32.

[15] N. Murata, *A statistical study of on-line learning*, in On-line Learning in Neural Networks, Cambridge University Press, Cambridge, UK, 1998, pp. 63–92.

[16] A. Nedić and D. Bertsekas, *Convergence rate of incremental subgradient algorithms*, in Stochastic Optimization: Algorithms and Applications, Springer, Boston, 2001, pp. 223–264.

[17] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, *Robust stochastic approximation approach to stochastic programming*, SIAM J. Optim., 19 (2009), pp. 1574–1609.

[18] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., Springer New York, 1999.

[19] H. Park, S.-I. Amari, and K. Fukumizu, *Adaptive natural gradient learning algorithms for various stochastic models*, Neural Networks, 13 (2000), pp. 755–764.

[20] A. Plakhov and P. Cruz, *A stochastic approximation algorithm with step-size adaptation*, J. Math. Sci., 120 (2004), pp. 964–973.

[21] M. J. D. Powell, *Some global convergence properties of a variable metric algorithm for minimization without exact line searches*, in Nonlinear Programming, R. W. Cottle and C. E. Lemke, eds., SIAM-AMS Proc. 9, AMS, Providence, RI, 1976, pp. 53–72.

[22] H. Robbins and S. Monro, *A stochastic approximation method*, Ann. Math. Statist., 22 (1951), pp. 400–407.

[23] N. L. Roux and A. W. Fitzgibbon, *A fast natural Newton method*, in Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 623–630.

[24] N. L. Roux, P.-A. Manzagol, and Y. Bengio, *Topmoumoute online natural gradient algorithm*, in Advances in Neural Information Processing Systems 20, MIT Press, Cambridge, MA, 2007, pp. 849–856.

[25] N. Schraudolph, J. Yu, and S. Günter, *A stochastic quasi-Newton method for online convex optimization*, in Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, Microtome Publishing, Brookline, MA, 2007, pp. 436–443.

[26] P. Sunehag, J. Trumpf, S. V. N. Vishwanathan, and N. Schraudolph, *Variable metric stochastic approximation theory*, in Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, Microtome Publishing, Brookline, MA, (2007), pp. 436–443.

[27] P. Tseng and S. Yun, *A coordinate gradient descent method for nonsmooth separable minimization*, Math. Program., 117 (2009), pp. 387–423.

[28] F. Yousefian, A. Nedić, and U. V. Shanbhag, *On stochastic gradient and subgradient methods with adaptive steplength sequences*, Automatica J. IFAC, 48 (2012), pp. 56–67.