Open access • Journal Article • DOI:10.1137/10079923X

On the use of stochastic hessian information in optimization methods for machine learning — Source link

Richard H. Byrd, Gillian M. Chin, Will Neveitt, Jorge Nocedal

Institutions: University of Colorado Boulder, Northwestern University, Google

Published on: 22 Sep 2011 - Siam Journal on Optimization (Society for Industrial and Applied Mathematics)

Topics: Online machine learning, Stability (learning theory), Active learning (machine learning), Hessian matrix and Broyden–Fletcher–Goldfarb–Shanno algorithm

On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning

Richard H. Byrd* Gillian M. Chin† Will Neveitt ‡ Jorge Nocedal §

January 13, 2011

Abstract

This paper describes how to incorporate sampled curvature information in a Newton-CG method and in a limited memory quasi-Newton method for statistical learning. The motivation for this work stems from supervised machine learning applications involving a very large number of training points. We follow a batch approach, also known in the stochastic optimization literature as a sample average approximation (SAA) approach. Curvature information is incorporated in two *sub-sampled Hessian algorithms*, one based on a matrix-free inexact Newton iteration and one on a preconditioned limited memory BFGS iteration. A crucial feature of our technique is that Hessian-vector multiplications are carried out with a significantly smaller sample size than is used for the function and gradient. The efficiency of the proposed methods is illustrated using a machine learning application involving speech recognition.

*Department of Computer Science, University of Colorado, Boulder, CO, USA. This author was supported by National Science Foundation grant CMMI 0728190 and Department of Energy grant DE-SC0001774.

†Department of Industrial Engineering and Management Sciences, Northwestern University. This author was supported by an NSERC fellowship and a grant from Google Inc.

‡Google Research.

§Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA. This author was supported by National Science Foundation grant DMS-0810213 and by Department of Energy grant DE-FG02-87ER25047-A004.

1 Introduction

The inexact Newton-CG method and the limited memory BFGS method are useful techniques for solving large-scale deterministic optimization problems in which the function and the gradient can be computed with good accuracy but the Hessian matrix cannot be formed or factored at reasonable cost [11]. In this paper, we consider optimization problems whose objective function is given by an expectation or a loss function and can therefore be considered stochastic. We ask whether one can design Newton-CG and limited memory BFGS methods that exploit the properties of such objective functions. In this paper, we propose to employ sampled (or stochastic) curvature information, using a small batch size, to accelerate these optimization methods.

The motivation for this work stems from supervised machine learning, where the goal is to estimate a statistical model on training data that makes accurate predictions on previously unseen data. Given a random set of training points (x_i, y_i) drawn from some probability distribution, and given a loss function $\ell(w; y_i, x_i)$ parametrized by a vector $w \in \mathbb{R}^n$, we seek to find the vector w that minimizes the expected loss. The optimization problem can thus be stated as

$$\min_{w \in \mathbb{R}^n} \quad J(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(w; y_i, x_i), \tag{1.1}$$

where $x_i \in \mathbb{R}^{\mathrm{NF}}$ represent the feature vectors of the training points and $y_i \in \mathbb{R}^{\mathrm{NC}}$ their corresponding labels (or classes). The right hand side in (1.1) represents an expectation taken over the random set of training points. We are mainly interested in the case when the loss function $\ell(\cdot, x_i, y_i)$ is smooth and convex, as is the case in log-linear models for entropy maximization. We are motivated by applications, such as speech recognition, where the number of training points, m, is exceedingly large (in the millions or even billions) and the number of variables, w, is large (in the tens of thousands or millions). The evaluation of the objective function $J(w)$ is therefore very costly in these applications, and determining an exact solution by standard optimization methods is very time consuming.

In order to reduce the computational cost of the optimization, and given that the training set is often highly redundant, it is common to consider only a random sample of the training points, i.e., to include only a subset of the summation terms in (1.1) in the optimization process, thus following a *sample average approximation* (SAA) framework. If we define $\mathcal{D} = \{1, 2, \cdots, m\}$ and let $\mathcal{X} \subseteq \mathcal{D}$ be a random sample consisting of $|\mathcal{X}|$ training instances $(y_i, x_i)_{i \in \mathcal{X}}$, we can define a stochastic approximation of the true objective $J(w)$ as

$$J_{\mathcal{X}}(w) = \tfrac{1}{|\mathcal{X}|} \sum_{i \in \mathcal{X}} \ell(w; y_i, x_i). \tag{1.2}$$

If the sample $\mathcal{X}$ is large enough (a so-called batch or SAA approach [13]), we can apply a conventional gradient-based method to minimize $J_{\mathcal{X}}$. If on the other hand, the sample $\mathcal{X}$ is very small (the on-line or SA approach), we can apply one step of a stochastic gradient method [12, 1], choose another small sample $\mathcal{X}$, and repeat the process.

The two methods proposed in this paper are designed for batch (or mini-batch) applications – not for online settings where the optimization is performed using highly inaccurate

function and gradient information. For the applications we have in mind, one can use either the complete training set to define the loss function, as in (1.1), or only a subset of it (a mini-batch) as in (1.2). Our approach is based on the fact that less accuracy in the Hessian is required than in the gradient, and on the observation that, although the Hessian of $J_{\mathcal{X}}$ is typically very large and dense, one can compute products of a *sample* of the Hessian times vector terms at modest cost. Specifically, suppose that we have determined (by some consideration) that an appropriate sample for computing the function and gradient is $\mathcal{X}$. Then, we select a random subset of $\mathcal{X}$, which we denote as $\mathcal{S}$, and compute Hessian-vector products by including only those terms corresponding to $\mathcal{S}$.

In the Newton-CG method, we incorporate sampled (or stochastic) curvature information through a matrix-free conjugate gradient (CG) iteration applied to the Newton equations. We implement this idea by using a subsample that is much smaller than that used for the evaluation of the objective function $J_{\mathcal{X}}$ and its gradient $\nabla J_{\mathcal{X}}$, in the computation of the Hessian-vector products required by the CG iteration. By coordinating the size of the subsample and the number of CG iterations, the computational cost of this Newton-like iteration is comparable to the cost of a steepest descent step – but the resulting iteration is much more rapidly convergent. We refer to this algorithm as a *sub-sampled Hessian Newton method*.

In the limited memory BFGS (L-BFGS) method, we incorporate stochastic Hessian information through the so-called "initial matrix" employed in limited memory BFGS updating. In the standard L-BFGS method [6], this initial matrix is chosen at every iteration as a multiple of the identify matrix. In the proposed algorithm, the initial matrix is defined *implicitly* via a conjugate gradient solve of a linear system whose coefficient matrix is given by the stochastic Hessian. We call this technique the *stochastically initialized L-BFGS* method, and similarly to the approach described above, it is crucial that the stochastic curvature information provided to the algorithm uses a much smaller sample than that used for the evaluation of the objective function and its gradient.

The main goal of this paper is to propose that the use of approximate second derivative information of this kind can be useful in some machine learning applications. Additional gains in efficiency can be obtained in both methods by implementing them in a dynamic setting, where the sample $\mathcal{X}$ is initially small, and is increased automatically, as needed, so as to achieve any desired level of accuracy. This dynamic framework is, however, not explored here, as it is the subject of a future study.

An important motivation for the approach proposed in this paper stems from the availability of large-scale distributed computing environments that permit the parallel evaluation of the objective function (1.2) and obviate the need for working with very small batch sizes. In such a setting, $J_{\mathcal{X}}$, $\nabla J_{\mathcal{X}}$ and Hessian-vector products can all be evaluated in parallel by assigning subsets of the summation in (1.2) to different computing nodes. It is advantageous that the subsets are not too small, so that the latency in communication does not dominate the total computing time. Thus, the overall sample $\mathcal{X}$ need not be very small and the use of deterministic optimization techniques is justified.

A Newton-like method for machine learning has been studied in [5] in the context of classification problems involving 2 classes and a sparse set of features. Other Newton-like

methods and quasi-Newton methods for machine learning are discussed in [1, 4]. All of these approaches are significantly different from the techniques proposed here.

The paper is organized into 5 sections. In Section 2 we present the sub-sampled Hessian Newton-CG method and in Section 3, the L-BFGS variant. Numerical results on a speech recognition problem are reported in Section 4, and in Section 5 we provide some concluding remarks and open questions.

Notation. We follow the convention used in machine learning and denote the variables of the optimization problem (i.e. the parameters to be estimated) by w. Throughout the paper $\|\cdot\|$ denotes Euclidean vector norm.

2 The Sub-Sampled Hessian Newton Method

Let us begin by reviewing the Newton-CG method for unconstrained optimization (also known as the truncated or inexact Newton method; see e.g. [16, 17, 10]). The problem under consideration is to minimize a function $J : \mathbb{R}^n \to \mathbb{R}$.

At an iterate w_k, we apply the conjugate gradient (CG) method to compute an approximate solution p_k of the linear system

$$\nabla^2 J(w_k)p = -\nabla J(w_k). \tag{2.1}$$

The CG iteration is terminated when the residual $r_k = \nabla^2 J(w_k)p_k + \nabla J(w_k)$ is sufficiently small, or when a prescribed number of CG iterations have been performed. The new iterate is then given by $w_{k+1} = w_k + \alpha_k p_k$, where α_k is a steplength that ensures sufficient decrease in the objective function.

The conjugate gradient method does not require explicit knowledge of the Hessian matrix, but only requires products of this matrix times vectors. Therefore, we can implement the Newton-CG method in a matrix-free setting, provided we have the ability to compute these Hessian-vector products without forming the Hessian. This method is quite flexible: by controlling the number of CG iterations, it can resemble the steepest descent method, at one extreme, or the classical (exact) Newton method at the other extreme. However, if an effective preconditioner is not available, the Newton-CG method can be expensive because each Hessian-vector product is at least as costly as one gradient evaluation. In other words, the savings in outer iterations achieved by the Newton-CG method, normally do not compensate for the higher cost of the iteration.

We propose that for expectation minimization problems of the form (1.2), an effective way of implementing the Newton-CG method is to reduce the cost of the iteration by employing a smaller sample $\mathcal{S}$, and including only those terms corresponding to $\mathcal{S}$ in the computation of Hessian-vector products. In Section 4.1 we give an illustrative example in which the cost of the Hessian-vector products decreases linearly as the sample size decreases.

The method described in this section is motivated by the following three considerations: a) The stochastic nature of the objective (1.2) suggests that a natural way of incorporating curvature information into a Newton-like method inexpensively is to sample the Hessian;

b) Newton-like methods are much more tolerant to errors in the Hessian than in the computation of the function and its gradient, and therefore, one can use small samples for the representation of curvature information; c) instead of constructing Hessian approximations, once can incorporate curvature information through Hessian-vector products.

The Hessian subsample $\mathcal{S}$ can be chosen small enough so that the total cost of the CG iteration is not much larger than the cost of one gradient evaluation. On the other hand, $\mathcal{S}$ should be large enough so that the curvature information obtained through these Hessian-vector products is useful. One of the challenges in this approach is to achieve the right balance between these two goals.

The proposed algorithm is stated below. We recall that, given any sample $\mathcal{X}_k \subset \mathcal{D} = \{1, 2, ..., m\}$, the stochastic approximation $J_{\mathcal{X}_k}$ is defined by (1.2).

Algorithm S-Newton: Sub-Sampled Hessian Newton-CG Method

Choose an initial iterate w_0, constants $\eta, \sigma \in (0, 1)$, a CG iteration limit $\max_{cg}$, and initial samples $\mathcal{X}_0$ and $\mathcal{S}_0 \neq \emptyset$ such that $|\mathcal{S}_0| < |\mathcal{X}_0|$.

For $k = 0, 1, ...,$ until a convergence test is satisfied:

1. Evaluate $J_{\mathcal{X}_k}(w_k)$ and $\nabla J_{\mathcal{X}_k}(w_k)$.

2. Apply the matrix-free conjugate gradient method to compute an approximate solution p_k of the linear system
$$\nabla^2 J_{\mathcal{S}_k}(w_k)p = -\nabla J_{\mathcal{X}_k}(w_k). \tag{2.2}$$
The CG iteration is terminated when either $\max_{cg}$ iterations have been performed or when the residual $r_k = \nabla^2 J_{\mathcal{S}_k}(w_k)p_k + \nabla J_{\mathcal{X}_k}(w_k)$ satisfies
$$r_k \leq \sigma \|\nabla J_{\mathcal{X}_k}(w_k)\|. \tag{2.3}$$

3. Update the variables:
$$w_{k+1} = w_k + \alpha_k p_k, \tag{2.4}$$
where the steplength α_k is the largest element in the set $\{1, 1/2, 1/4, \cdots\}$ such that
$$J_{\mathcal{X}_k}(w_{k+1}) \leq J_{\mathcal{X}_k}(w_k) + \eta\alpha_k \nabla J_{\mathcal{X}_k}(w_k)^T p_k. \tag{2.5}$$

4. Choose new samples $\mathcal{X}_{k+1}$, $\mathcal{S}_{k+1}$ such that $|\mathcal{S}_{k+1}| < |\mathcal{X}_{k+1}|$.

The curvature information obtained in this manner can be expected to be useful in some statistical learning applications because, as mentioned above, Newton-like methods are both very tolerant to the choice of Hessian and can make good use of limited curvature information. Specifically, if B is any symmetric and positive definite matrix and if we apply *any* number of CG steps to the system $Bd = -\nabla J(w_k)$, the resulting Newton-CG step is a descent direction for $J(w_k)$; see section 2.1. In the machine learning applications that motivated this study, the (logistic) loss function ℓ in (1.2) is convex, and hence $\nabla^2 J_{\mathcal{S}}$ will be positive semi-definite for any non-empty choice of $\mathcal{S}$.

We have not specified in Algorithm S-Newton whether the sizes of the samples $\mathcal{X}_k$, $\mathcal{S}_k$ change at every iteration, or are kept fixed. The algorithm has been stated in sufficient generality to allow many strategies, including the "semi-stochastic" case when $\mathcal{S}_k \subset \mathcal{X}_k = \mathcal{D} = \{1, 2, \cdots, m\}$ for all k. Comparatively, Algorithm S-Newton is also capable of incorporating dynamic techniques in which the samples sizes $|\mathcal{S}_k| < |\mathcal{X}_k|$ are initially small (to benefit from the initial efficiency of stochastic gradient-type methods) and increase as needed to achieve the desired objective value. Regardless of the strategy chosen, and in order to avoid bias, the subsample $\mathcal{S}_k$ should be recomputed at every (outer) iteration of the sub-sampled Hessian Newton method. Thus, even if the size of the sample $\mathcal{S}_k$ remains constant throughout the iteration, the sample itself should change, and would typically be chosen as a subset of $\mathcal{X}_k$. For the sake of simplicity, in this paper, we analyze and test only the semi-stochastic case $\mathcal{S}_k \subset \mathcal{X}_k = \mathcal{D}$ and defer the study of dynamic sampling strategies to a future study.

Let us quantify the cost of the search direction computation in the sub-sampled Hessian Newton method. Let g_{cost} denote the cost of computing the gradient $\nabla J_{\mathcal{X}_k}$ and $\max_{cg}$ the maximum number of CG iterations permitted. Suppose that the cost of one Hessian-vector product is $factor \times g_{cost}$. Then, assuming that the maximum limit of CG iterations is always reached, the cost of the step computation in Algorithm S-Newton (excluding the function and gradient evaluation) is given by

$$\max_{cg} \times factor \times g_{cost}.$$

In the deterministic Newton-CG method, which corresponds in our case to the choice $\mathcal{S}_k = \mathcal{X}_k = \mathcal{D}$, we have that *factor* is at least 1, and $\max_{cg}$ can range from 5 to several dozen. Thus, the cost of one iteration of the classical Newton-CG method can easily be 10 times higher than the cost of a gradient computation, and this causes the method to be less competitive than limited memory quasi-Newton and nonlinear CG methods on many problems.

Now, by decreasing the sample size $\mathcal{S}$, we can reduce the ratio *factor* significantly. For example, if $|\mathcal{S}_k|$ is one tenth of $|\mathcal{X}_k|$, then *factor* will be about one tenth. In general, we can coordinate the size of the sample size $|\mathcal{S}_k|$ with the maximum allowable number of CG iterations so that

$$factor \times \max_{cg} \approx 1,$$

and thus the total cost of the step computation is comparable to the cost of one gradient evaluation. The freedom in the selection of the subsample $\mathcal{S}_k$ thus provides the sub-sampled Hessian Newton-CG method with much flexibility. We should also point out that for the solution of the subsampled Hessian Newton equation (2.2), scale invariance is preserved with respect to changes in scale of w.

2.1 Convergence Properties

Based on some well-known properties of the CG method, it is easy to show that the semi-stochastic version of the Newton method (with $\mathcal{X}_k = \mathcal{D}$) is globally convergent on problems of the form (1.1), provided the subsampled Hessians are uniformly positive definite, i.e.,

there is a constant $\gamma_1 > 0$ such that for all k and all $v \in \mathbb{R}^n$

$$v^T \nabla^2 J_{\mathcal{S}_k}(w_k) v \geq \gamma_1 \|v\|. \tag{2.6}$$

For objective functions of the form (1.1) the loss term ℓ is often convex (although not strongly so) but J can be made uniformly convex by adding a regularization term of the form $\delta\|w\|^2$ to the right hand side of (1.1). Furthermore, uniform convexity of J implies that the sequence of iterates w_k is bounded, and by continuity of $\nabla^2 J$ there is a constant γ_2 such that, for all k and all $\mathcal{S}_k$,

$$\|\nabla^2 J_{\mathcal{S}_k}(w_k)\| \leq \gamma_2. \tag{2.7}$$

For the following discussion, we denote $J \triangleq J_{\mathcal{D}}$.

Theorem 2.1 *Let $J_{\mathcal{D}} : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable and uniformly convex and suppose that condition* (2.6) *holds. Then, the sequence of iterates $\{w_k\}$ generated by Algorithm S-Newton, with $\mathcal{X}_k = D$, satisfies*

$$\lim_{k\to\infty} \nabla J_{\mathcal{D}}(w_k) = 0. \tag{2.8}$$

Proof. First we show that the search directions p_k obtained by applying *any* number of CG steps to the system (2.2) are directions of strong descent for $J_{\mathcal{D}}(w_k)$.

It is a well known fact [7] that the iterates generated by the CG method applied to the system (2.2) minimize the quadratic function

$$\tfrac{1}{2} p^T \nabla^2 J_{\mathcal{S}_k}(w_k) p + p^T \nabla J_{\mathcal{D}}(w_k)$$

over a Krylov subspace that includes the vector $\nabla J_{\mathcal{D}}(w_k)$. Let us define Q to be an orthonormal basis for this space. Then, the search direction p_k can be expressed as $p_k = Qv$, for some vector v satisfying

$$[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q] v = -Q^T \nabla J_{\mathcal{D}}(w_k). \tag{2.9}$$

Since $\nabla J_{\mathcal{D}}(w_k)$ is in the range of Q (it is in the Krylov space mentioned above), we have that $\|Q^T \nabla J_{\mathcal{D}}(w_k)\| = \|\nabla J_{\mathcal{D}}(w_k)\|$, and hence by the orthogonality of Q,

$$\begin{aligned}
\nabla J_{\mathcal{D}}(w_k)^T p_k &= -\nabla J_{\mathcal{D}}(w_k)^T Q [Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]^{-1} Q^T \nabla J_{\mathcal{D}}(w_k) && (2.10)\\
&\leq -\frac{\|Q^T \nabla J_{\mathcal{D}}(w_k)\|^2}{\|Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q\|} && (2.11)\\
&\leq -\frac{\|\nabla J_{\mathcal{D}}(w_k)\|^2}{\|\nabla^2 J_{\mathcal{S}_k}(w_k)\|} && (2.12)\\
&\leq -\|\nabla J_{\mathcal{D}}(w_k)\|^2/\gamma_2, && (2.13)
\end{aligned}$$

where γ_2 is defined in (2.7). In addition, we have from (2.9) and (2.6) that

$$\|p\| = \|Q[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]^{-1} Q^T \nabla J_{\mathcal{D}}(w_k)\| \leq \|\nabla J_{\mathcal{D}}(w_k)\|/\gamma_1.$$

It follows that

$$-\frac{p^T \nabla J_{\mathcal{D}}(w_k)}{\|\nabla J_{\mathcal{D}}(w_k)\|\|p\|} \geq \gamma_1/\gamma_2,$$

which proves that p_k is a strong direction of descent for $J_{\mathcal{D}}$ at w_k. We also have from (2.9) and (2.7) that

$$\|\nabla J_{\mathcal{D}}(w_k)\| = \|Q^T \nabla J_{\mathcal{D}}(w_k)\| \leq \|\nabla^2 J_{\mathcal{S}_k}(w_k)\|\|v\| \leq \gamma_2 \|v\|,$$

so that

$$\|p_k\| = \|v\| \geq \|\nabla J_{\mathcal{D}}(w_k)\|/\gamma_2.$$

We can now apply Zoutendijk's classical analysis to prove global convergence. Specifically, we have shown that all the conditions in Theorem 11.7 in ([3, p.379]) are satisfied and it follows that the limit (2.8) holds. □

3 Stochastically Initialized L-BFGS Method

The limited memory BFGS method (L-BFGS) [11] maintains very simple approximations of the Hessian of $J(w)$. Curvature information from gradients at a few recent iterations is used to construct a Hessian approximation in a way that does not store a fully dense n by n matrix but is parsimonious in terms of computational time and memory space.

In the standard BFGS method, an approximation H_k of the inverse Hessian $\nabla^2 J(w_k)^{-1}$ is updated at every iteration so that the secant equation is satisfied at each step, i.e.,

$$H_{k+1} y_k = s_k \qquad \text{where} \qquad y_k = \nabla J(w_{k+1}) - \nabla J(w_k), \qquad s_k = w_{k+1} - w_k. \tag{3.1}$$

The BFGS update formula (see, e.g. Fletcher [2]) is given by:

$$H_{k+1} = (I - \rho_k y_k s_k^T)^T H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \qquad \text{where } \rho_k = 1/y_k^T s_k. \tag{3.2}$$

The L-BFGS method does not construct the Hessian approximation explicitly, but defines it indirectly based on stored secant information from the t most recent iterations, where t is a small integer. If we define $V_k = (I - \rho_k y_k s_k^T)$, apply the BFGS formula t times, and expand the resulting equation to encompass all computations from an initial approximation H_k^0, we obtain

$$\begin{aligned} H_k = {} & (V_{k-1}^T V_{k-2}^T \cdots V_{k-t}^T) H_k^0 (V_{k-t} V_{k-t+1} \cdots V_{k-1}) \\ & + \rho_{k-t} (V_{k-1}^T \cdots V_{k-t+1}^T) s_{k-t} s_{k-t}^T (V_{k-t+1} \cdots V_{k-1}) \\ & + \rho_{k-t+1} (V_{k-1}^T \cdots V_{k-t+2}^T) s_{k-t+1} s_{k-t+1}^T (V_{k-t+2} \cdots V_{k-1}) \\ & + \cdots \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned} \tag{3.3}$$

The search direction of the L-BFGS method is defined as

$$p_k = -H_k \nabla J(w_k). \tag{3.4}$$

Rather than forming the matrices H_k, one can store the correction pairs $\{s_i, y_i\}$ that define them, and compute the product $H_k \nabla J(w_k)$ via the relation (3.3). This matrix-vector multiplication can be carried out very efficiently by the two loop recursion described in [11, p.112] at a cost of about $4tn$ multiplications plus the cost of one multiplication by H_k^0.

In the standard L-BFGS method, the so-called *initial matrix* H_k^0 in (3.3) is defined afresh at every iteration, with a common choice being

$$H_k^0 = \gamma_k I \qquad \text{where} \qquad \gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \tag{3.5}$$

Such a choice of H_k^0 contains little curvature information about the problem. Therefore we ask whether it is possible to define H_k^0 in a way that exploits the structure of a loss function of the form (1.1).

We propose to define H_k^0 *implicitly* through the use of a conjugate gradient iteration. To see how this can be done, note that when performing the computation of $H_k \nabla J(w_k)$ through (3.3) we must compute, at some point, a product of the form

$$r \leftarrow H_k^0 q, \qquad \text{for some vector } q \in \mathbb{R}^n.$$

Ideally, H_k^0 would be defined as an accurate approximation of the inverse of the Hessian, i.e. $H_k^0 \approx \nabla^2 J(w_k)^{-1}$. Therefore, in the ideal scenario we would compute r as the solution of the linear system

$$\nabla^2 J(w_k) r = q. \tag{3.6}$$

Instead of solving this system exactly, we could perform only a few iterations of the matrix-free conjugate gradient method, and define the vector r as the resulting approximate solution of (3.6). This approach therefore eliminates the need for specifying the initial matrix, and implicitly defines H_k^0 through an approximate matrix-free conjugate gradient solve of the system (3.6).

As in the Newton method of the previous section, we employ a smaller sample to define the Hessian in (3.6), compared to the sample used for the function and gradient computation so that the cost of Hessian-vector products within the CG method is affordable. Thus, given a sample $\mathcal{S}_k$ such that $|\mathcal{S}_k| < |\mathcal{X}_k|$, we define the vector r as an approximate solution to the system

$$\nabla^2 J_{\mathcal{S}_k}(w_k) r = q \tag{3.7}$$

computed by the conjugate gradient method. The computation of the product $H_k \nabla J_{\mathcal{X}_k}(w_k)$ in this semi-stochastic L-BFGS approach can be stated as follows (c.f.[11, p.112]):

Procedure I: Two-loop Recursion with Implicit Stochastic Initial Matrix
$q \leftarrow \nabla J_{\mathcal{X}_k}(w_k)$
for $i = k-1, k-2, \cdots, k-t$
  $\alpha_i \leftarrow \rho_i s_i^T q$;
  $q = q - \alpha_i y_i$;
end(for)
$r \leftarrow$ approximate solution to (3.7) obtained by the matrix-free **CG Method** ;
for $i = k-t, k-t+1, \cdots, k-1$
  $\beta \leftarrow \rho_i y_i^T r$;
  $r \leftarrow r + s_i(\alpha_i - \beta)$;
end(for)
STOP: result $r = H_k \nabla J_{\mathcal{X}_k}(w_k)$

We terminate the CG method when either the residual condition (2.3) is met, or when a CG iteration limit is reached. The precise description of the stochastic L-BFGS method is as follows.

Algorithm SLM: Stochastically Initialized L-BFGS Algorithm
Choose an initial iterate w_0, a CG iteration limit $\max_{cg}$, initial samples $\mathcal{X}_0$ and $\mathcal{S}_0 \neq \emptyset$ such that $|\mathcal{S}_0| < |\mathcal{X}_0|$, and constants $0 < c_1 < c_2 < 1$. Set $k \leftarrow 0$.

1. Evaluate $J_{\mathcal{X}_0}(w_0)$ and $\nabla J_{\mathcal{X}_0}(w_0)$.
   Set $p_0 = -\nabla J_{\mathcal{X}_0}(w_0)$
2. **While:** Convergence Test is not satisfied:
   2.1 Store: w_k and $\nabla J_{\mathcal{X}_k}(w_k)$.
   2.2 Line Search: compute steplength α_k that satisfies the Wolfe conditions
      1. $J_{\mathcal{X}_k}(w_k + \alpha_k p_k) \leq J_{\mathcal{X}_k}(w_k) + c_1 \alpha_k \nabla J_{\mathcal{X}_k}(w_k)^T p_k$
      2. $\nabla J_{\mathcal{X}_k}(w_k + \alpha_k p_k)^T p_k \geq c_2 \nabla J_{\mathcal{X}_k}(w_k)^T p_k$.
   2.3 Compute new iterate: $w_{k+1} \leftarrow w_k + \alpha_k p_k$.
   2.4 Update $s_k \leftarrow w_{k+1} - w_k$ and $y_k \leftarrow \nabla J_{\mathcal{X}_k}(w_{k+1}) - \nabla J_{\mathcal{X}_k}(w_k)$.
   2.5 Set $k \leftarrow k+1$.
   2.5 Re-sample $\mathcal{X}_k$, $\mathcal{S}_k$ such that $|\mathcal{S}_k| < |\mathcal{X}_k|$.
   2.6 Evaluate $J_{\mathcal{X}_k}(w_k)$ and $\nabla J_{\mathcal{X}_k}(w_k)$.
   2.7 Compute direction vector p_k using **Procedure I**.
3. **End(While)**

A variety of strategies can be employed to choose the samples $\mathcal{X}_k$, $\mathcal{S}_k$ at every iteration. For simplicity, our testing is done for the semi-stochastic case $\mathcal{S}_k \subset \mathcal{X}_k = \mathcal{D}$. As before,

to avoid bias, the subsample $\mathcal{S}_k$ is recomputed at every iteration of the stochastic L-BFGS algorithm; see section 4.

We now show that Algorithm SLM is globally convergent on convex problems.

Theorem 3.1 *Under the assumptions on J given in Theorem 2.1, the sequence of iterates $\{w_k\}$ generated by Algorithm SLM satisfies*

$$\lim_{k\to\infty} \nabla J_{\mathcal{D}}(w_k) = 0. \tag{3.8}$$

Proof. It has been shown in [6] that the L-BFGS algorithm is globally convergent on convex problems provided the symmetric positive definite initial matrices H_k^0 have eigenvalues that are uniformly bounded above and away from zero. In Algorithm SLM, these matrices are not constructed explicitly; instead we compute the vector $r \leftarrow H_k^0 q$ via the approximate CG solution of (3.7). Thus, we need to show that at every iteration, the vector r computed in this manner can be expressed as the product of a matrix (with the desired properties) times q.

Since the vector r is the result of, say j, CG steps applied to a linear system (3.7), it follows from the same argument that lead to the equation (2.9) that $r = Qv$, where

$$[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]v = Q^T q$$

and the columns of Q are an orthonormal basis for the Krylov subspace generated by the CG iteration. Thus,

$$r = (Q[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]^{-1} Q^T) q.$$

The matrix multiplying q is singular. However, q is in the Krylov subspace (and hence in the range of Q) and therefore we also have that

$$r = (Q[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]^{-1} Q^T + \bar{Q}\bar{Q}^T) q \stackrel{\text{def}}{=} H_k^0 q, \tag{3.9}$$

where $\bar{Q}$ is a $n \times (n-j)$ matrix whose columns are an orthonormal basis for the null space of Q^T, so that $\bar{Q}^T q = 0$. The matrix inside the square brackets can therefore be regarded as the initial matrix, H_k^0, for the limited memory update (3.3). The eigenvalues of H_k^0 are as follows: $n-j$ are equal to 1 and correspond to the eigenvalues of $\bar{Q}\bar{Q}^T$, and the rest are given by the j nonzero eigenvalues of $Q[Q^T \nabla^2 J_{\mathcal{S}_k}(w_k) Q]^{-1} Q^T$. By the orthogonality of Q these j eigenvalues are in the the interval $[1/\gamma_2, 1/\gamma_1]$ where γ_1 and γ_2 are defined in (2.6) and (2.7). Therefore, the eigenvalues of the matrix H_k^0 defined in (3.9) are all in the interval $[\min\{1, 1/\gamma_2\}, \max\{1, 1/\gamma_1\}]$.

It then follows from Theorem 7.1 of Liu and Nocedal [6] that the limit (3.8) is satisfied. □

3.1 Numerical Behavior in the Deterministic Setting

Before testing Algorithm SLM on statistical learning problems of the form (1.1), let us verify that the incorporation of curvature information, as described above, is beneficial in the ideal case when the objective function is deterministic and $\mathcal{S}_k = \mathcal{X}_k = \mathcal{D}$, i.e. when the matrix-vector products employed in the CG solve of (3.7) use the exact Hessian. This will serve as indication that the approach is not unsound.

Test 1: Quadratic Function. We consider the problem of minimizing the convex quadratic

$$f(w) = \sum_{j=1}^{100} (100 - j + 1) w_j^2,$$

whose Hessian has a condition number of 100. In Table 1 we report the number of iterations and function evaluations required by Algorithm SLM as a function of the maximum allowed number of CG iterations ($\max_{cg}$). In the bottom row, we report the performance of the standard L-BFGS method.

Table 1: Results on a Quadratic Function

$\max_{cg}$	SLM Iter	SLM Functions	Total CG Iter
1	95	96	94
5	13	14	60
10	8	9	70
15	6	7	73
20	5	6	74
L-BFGS	74	79	

Note from Table 1 that there is a consistent decrease in the number of iterations of Algorithm SLM as $\max_{cg}$ increases, showing the beneficial effect of incorporating curvature information through the initial matrix. The SLM method cannot, however, be considered successful in this deterministic setting (where $\mathcal{S}_k = \mathcal{X}_k$) because, as shown in the last column of Table 1, the total number of CG iterations is too expensive, even with the reduction observed in outer iterations.

Test 2: Quadratic Plus Exponential. To increase the complexity of the problem, we introduce an exponential term to the previous quadratic function, resulting in the convex function

$$f(w) = \sum_{j=1}^{100} \left((100 - j + 1) w_j^2 + e^{w_j} \right).$$

As in the previous example, we observe in Table 2 a steady reduction in the number of SLM iterations as the maximum allowable number of CG iteration ($\max_{cg}$) increases,

Table 2: Results on a Quadratic + Exponential Function

$\max_{cg}$	SLM Iter	SLM Functions	Total CG Iter
1	83	84	82
5	12	13	55
10	8	9	70
15	6	7	72
20	6	7	91
L-BFGS	66	70	

however the overall cost, as indicated by the last column, is too expensive to justify the savings in outer iterations.

These experiments therefore, suggest that the incorporation of curvature information in the initial matrix H_k^0 has a beneficial effect in terms of total outer iteration counts. In the next section, we show that by decreasing the cost of the CG iteration through stochastic Hessian sub-sampling we can make the SLM approach competitive in terms of computing time.

4 Numerical Tests

To assess the effectiveness of the sub-sampled Hessian methods proposed in this paper, we analyze and document their performance on a challenging machine learning problem involving multi-class classification of speech frames. Our benchmark is the standard L-BFGS algorithm, which is widely used in the machine learning community for tasks of this type [8, 9, 14]. As only the semi-stochastic form of the new methods is analyzed in this paper, we assume that $\mathcal{X}_k = \mathcal{D}$, and thus the function and gradient evaluations will use 100% of the information available for every iteration. The stochastic Hessian information will be based on a smaller sample $\mathcal{S}_k$ — typically 5% and 10% of the sample used for the function and gradient. For convenience, we define the *Hessian-vector sampling percentage* p%, as

$$p = \frac{|\mathcal{S}|}{|\mathcal{D}|}100. \tag{4.1}$$

To determine the Hessian samples $\mathcal{S}_k$, we first read in the training data and randomly shuffle it. Specifically, each training pair (x_i, y_i) is assigned a random index $i \in \{1, \ldots, \mathcal{D}\}$, excluding indices that have already been assigned. The array of indices obtained in this manner is divided into blocks in a sequential manner. Thus, the first iteration of the optimization algorithm will use the first p% of indices to define $\mathcal{S}_1$. At the second iteration, the next p% of the training points will be assigned to computing $\mathcal{S}_2$, and so on. If we reach the end of the array we wrap the current data block around to the beginning of the data set. Clearly, one could use a higher degree of randomization, but this will not have a noticeable effect on our algorithms.

4.1 A Speech Recognition Problem.

The objective of the speech recognition problem employed in our tests, is to construct a representative multinomial logistic regression model that maximizes the probability of correct classification amongst the data points included within the training set. Each training point consists of a real valued vector representing features for a 10 millisecond frame of speech, and a label representing the phonetic state assigned to that frame based on a human transcription of the source utterance. The training set was provided by Google. Our objective is to maximize the conditional probability of the correct phonetic state given the observed real valued vector. The variables and parameters of the problem are described as follows.

Parameters:

m : number of training points: $168,776$.

$\mathcal{C}$: the set of all class labels: $\{1, 2, \cdots, 128, 129\}$.

NF : number of feature measurements per data point: 79

y_h : the class label associated with data point $h : y_h \in \mathcal{C}$.

x_h : feature vector for data point h

$x_h(j)$: indicates the j^{th} entry of the feature vector

Decision Variable:

w : a parameter vector of dimension $|\mathcal{C}| \times NF = 10,191$;

w_i : parameter sub-vector for class label i;

$w_i(j)$: indicates the j^{th} entry of the parameter vector w_i.

Given pertinent feature information for a set of data points, the goal is to develop a model that will suggest the correct class label for each point, on average, with reasonably high probability. For each individual class label, the parameter vector w will have a value associated with each respective feature measurement. Thus, each class label will have a parameter vector of size NF, and as there are 129 different class labels and 79 pertinent feature measurements, the total number of decision variables for this speech problem is $129 \times 79 = 10,191$.

As we hope to build a statistical model that will suggest the correct class for each data point in the training set, an intuitive method for parameterizing the statistical model, based on observed data, is maximum likelihood estimation. By performing maximum likelihood estimation, we can not only determine the parameters of the statistical model that most likely represents the data, but also derive several attractive asymptotic properties for these values, such as consistency and efficiency. Therefore, in choosing the method of maximum likelihood estimation, we define the objective function J of the *minimization problem* (1.1)

as the normalized sum of the negative log likelihood of each data point being placed in the correct class, over all data points used in the training set:

$$J(w) = -\frac{1}{m}\sum_{h=1}^{m} \log \frac{\exp(w_{y_h}^T x_h)}{\sum_{i\in\mathcal{C}} \exp(w_i^T x_h)} \tag{4.2}$$
$$= \frac{1}{m}\Bigg[\sum_{h=1}^{m} \log \sum_{i\in\mathcal{C}} \exp(w_i^T x_h) - \sum_{h=1}^{m} \exp(w_{y_h}^T x_h)\Bigg].$$

The gradient can be written as follows,

$$\frac{\partial J}{\partial w_i(j)} = \frac{1}{m}\Bigg[\sum_{h=1}^{m} P(i,h)x_h(j) - \sum_{h:y_h=i}^{m} x_h(j)\Bigg], \tag{4.3}$$

where

$$P(i,h) = \left[\frac{\exp(w_i^T x_h)}{\sum_{j\in\mathcal{C}} \exp(w_j^T x_h)}\right].$$

Note that $P(i,h)$ is the designated probability of class label $i \in \mathcal{C}$ being the correct class label for data point h, given the parameter variable w.

The sub-sampling methods rely on the fact that Hessian-vector products can be computed efficiently. One can show that, given a vector $v \in \mathbb{R}^n$, if we divide it into segments initialized by class labels, and therefore $v_i \in \mathbb{R}^{NF}$ for $i \in \mathcal{C}$, we have

$$[\nabla^2 J(w)v]_{(i,j)} = \frac{1}{m}\sum_{h=1}^{m}\left(P(i,h)\ x_h(j)\left[v_i^T x_h - \frac{\sum_{j\in\mathcal{C}} v_j^T x_h \cdot exp(w_j^T x_h)}{\sum_{j\in\mathcal{C}} exp(w_j^T x_h)}\right]\right). \tag{4.4}$$

We have used the pair (i,j) to index the vector $\nabla^2 J(w)v$ (which is of dimension $|\mathcal{C}| \times NF$), where $i \in \mathcal{C}$ refers to the class and j denotes the feature (i.e. $j \in \{1, 2, \cdots, 79\}$). Thus, each entry of the Hessian-vector product is composed of a summation over the set of data points, and this presents an opportunity to incorporate second order information in a controlled manner such that the advantages of curvature information are balanced against the processing cost. Formula (4.4) represents a summation across all terms within the data set $\mathcal{D}$; to sample the Hessian, we select a subsample $\mathcal{S}_k$ as described above, and include only the corresponding terms in the summation.

4.2 Testing the Sub-Sampled Hessian Newton Method

In Figure 1, we compare the behavior of three methods:

i) The standard L-BFGS method [6] with memory $t = 20$

ii) The classical Newton-CG method (CN) [11, p.169] with $\max_{cg} = 10$ and full Hessian information, i.e with $p = 100\%$, where p is defined in (4.1).

iii) The sub-sampled Hessian Newton method (SN) (Algorithm S-Newton) with $\max_{cg} = 10$ and $p = 5\%$.

As function, gradient and Hessian-vector product evaluations are (by far) the most costly computations in the three algorithms tested, the computational effort in expressions (4.2), (4.3) and (4.4) will be used in lieu of CPU time. We refer to this computational effort as the "number of accessed points". To further define this term, we can observe that the objective function is a sum of terms, where each term is associated with a single training point. As a result, when computing the objective function, we can separate the overall function into smaller blocks, where each block is associated with a single training point. Therefore, all blocks will have identical wall times, and cumulatively represent the objective function when aggregated. Similar blocks appear in the gradient and Hessian-vector products, which in turn will have approximately the same wall-time as the blocks which collectively represent the function evaluation. We refer to the total number of computed blocks as the "number of accessed data points". In all figures in this paper, the horizontal axis plots this number. (Plots based on CPU time exhibit similar behavior as those presented.) The vertical axis plots the probability of correct classification, which is defined as $\exp(-J(w_k))$, with J given in (4.2).

We observe from the behavior presented in Figure 1 that the sub-sampled Hessian Newton method is the most efficient of all the methods for a correct classification probability greater than 0.08. Notably, for a probability level of 13% correct classification, the sub-sampled Newton method is approximately three times as fast as the classical Newton-CG method and twice as fast as the L-BFGS method. Note that the number of accessed points is of order 10^6 and the number of training points is about 168,776.

In Figure 2 we analyze the behavior of the sub-sampled Hessian Newton method as the CG iteration limit $\max_{cg}$ varies, while fixing the sampling percentage at $p = 5\%$. We report results for the settings of $\max_{cg} = 2, 5, 10, 50$. From these plotted graphs, we observe that the sub-sampled Hessian Newton method is not effective for $\max_{cg} = 2$, but is quite efficient for $\max_{cg} = 10$ and $\max_{cg} = 50$. This behavior is similar for higher sampling percentages, such as $p = 10\%$. The CG limit was reached at most iterations. Note that little is lost in terms of efficiency by truncating the conjugate gradient limit $\max_{cg}$ from 50 to 10. Given this observation, we fix the conjugate gradient limit to the value of $\max_{cg} = 10$, and report in Figure 3 the performance of the sub-sampled Hessian Newton method for varying Hessian-vector product sampling percentages p, specifically for $p = 1, 10, 50, 100\%$. We observe from Figure 3 that the sub-sampled Hessian Newton algorithm outperforms the benchmark L-BFGS algorithm for sampling percentages of $p = 1\%$ and 10%. Exhaustive testing indicates that, for this problem, the new algorithm is efficient for the range $p \in [1\%, 25\%]$.

4.3 Testing the Stochastically Initialized L-BFGS Method

In Figure 4, we compare the behavior of the standard L-BFGS method with memory $t = 5$ and the stochastically initialized L-BFGS method (Algorithm SLM) with $t = 5, \max_{cg} = 5$ and two choices of p, namely $p = 5\%, 100\%$. From this graph, it is observed that sub-sampling the Hessian vector products at the level $p = 5\%$, exhibits an increased acceleration

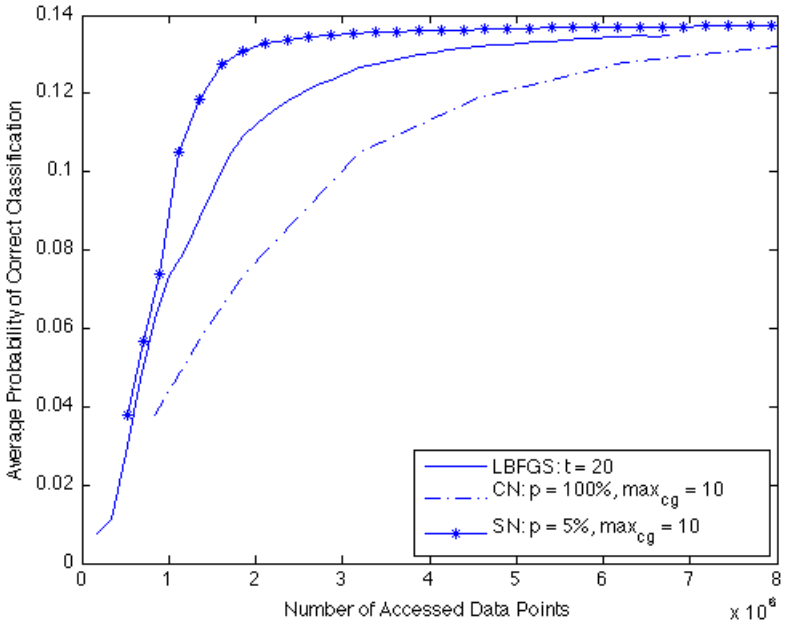

Figure 1: Comparison of L-BFGS vs. sub-sampled Hessian Newton (SN) vs. classical Newton (CN)

of performance over the traditional L-BFGS, especially near the convergent probability. More specifically, for an average probability classification of 13%, the reduction in number of data points required (or similarly, a reduction in CPU time) is approximately half of that required for L-BFGS. On the other hand, for the full sampling of Hessian-vector product, that is $p = 100\%$, the Algorithm SLM is significantly less efficient than the two other methods.

In order to analyze the effect of $\max_{cg}$ on Algorithm SLM, we fix $p = 5\%$ and report in Figure 5 results for the settings $\max_{cg} = 2, 5, 10$ and 50, in comparison to the standard L-BFGS algorithm with memory size $t = 5$. We observe from this figure that Algorithm SLM performs well for relatively small values for max_{cg}. However, unlike S-Newton, a large number of CG iterations is observed to be detrimental to the performance of the algorithm. Therefore, with a relatively small number of CG iterations below a certain threshold, observed to be $max_{cg} = 10$, the stochastically initialized L-BFGS method outperforms the standard L-BFGS algorithm.

To evaluate the behavior of Algorithm SLM for varying sampling percentages, p, we report in Figure 6 the results for $p = 1, 50, 100\%$, with $\max_{cg} = 10$ and memory setting $t = 5$. This figure clearly shows the gains in efficiency achieved by Algorithm SLM as the sampling percentage p decreases. For high percentages of sampling, which in turn leads to a larger $\mathcal{S}$ set, the worth of the information in the sample does not provide enough improvement

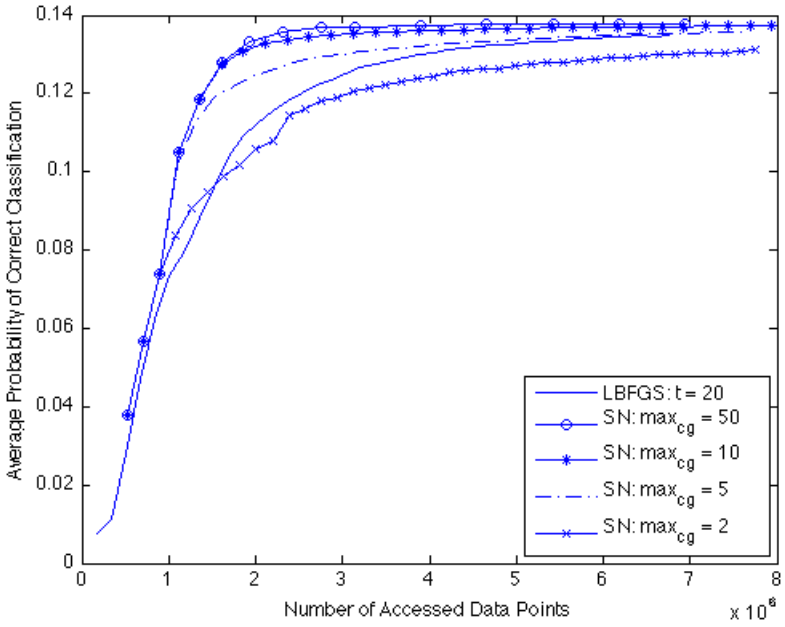

Figure 2: Behavior of the sub-sampled Hessian Newton method (SN), with $p = 5\%$, for varying $\max_{cg}$ values

over the benchmark L-BFGS algorithm to compensate for the additional processing time required. In comparison, for small sampling percentages such as $p = 1\%$, the balance between value of information and computing time contributes advantageously to Algorithm SLM, which is able to achieve substantial gains over standard L-BFGS.

4.4 Comparison of the Two Methods

In Figure 7 we compare the performance of the sub-sampled Hessian Newton method and the stochastically initialized L-BFGS method, where the sub-sample percentage of $p = 5\%$ is used. Both methods contain several parameters that affect their performance, and in Figure 7 we chose two settings for each method, being $max_{cg} = 5$ and 10. We see that the two approaches appear to be similar in terms of performance for this problem. In general, we feel that both algorithms can be effective for machine learning applications and the best choice among them may be problem dependent.

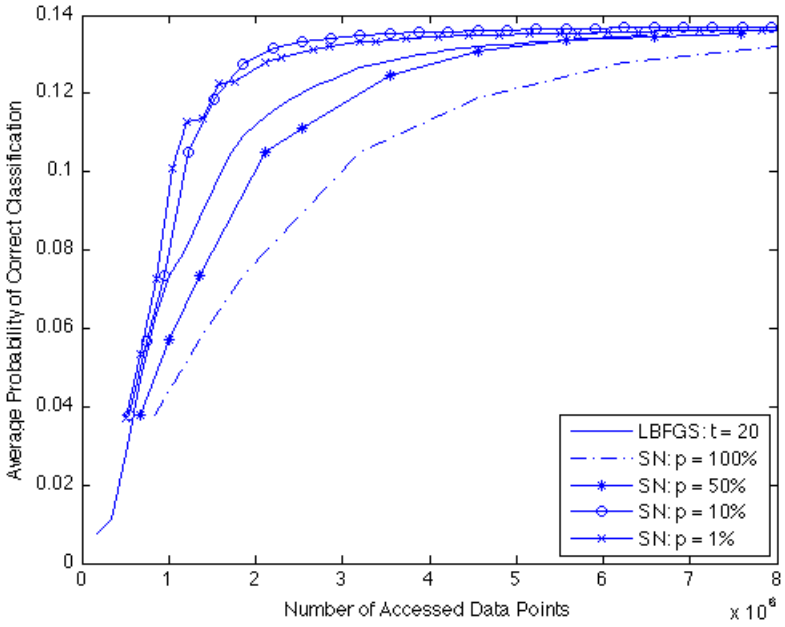

Figure 3: Behavior of the sub-sampled Hessian Newton method (SN), with $\max_{cg} = 10$, for varying values of the Hessian sub-sampling percentage p

5 Conclusions

We have proposed in this paper that Hessian sub-sampling via a matrix-free conjugate gradient iteration is an effective way of accelerating optimization methods for machine learning. Our method avoids sampling the second derivatives directly since this can lead to very noisy estimators, see [15]. We described two methods that can benefit from this approach, one is a variant of Newton-CG and the other of L-BFGS. There are a variety of ways of implementing these methods, depending on the choice of the function/gradient sample $\mathcal{X}_k$ and the Hessian subsample $\mathcal{S}_k$ at every iteration. In this paper, we have focused on the case when $\mathcal{X}_k$ is large; i.e. we have followed a batch (or SAA) approach. A key feature of our methods is that the Hessian subsample $\mathcal{S}_k$ is much smaller than $\mathcal{X}_k$.

The sub-sampled Hessian Newton-CG method overcomes one of the main drawbacks of inexact (or truncated) Newton methods, namely the high cost of computing a search direction, by significantly lowering the cost of the CG iteration. This is possible for stochastic objective functions of the form (1.2) because the computational cost of a Hessian-vector product decreases linearly with the sample size, and because small sample sizes provide useful curvature information. The stochastic Newton-CG method might be further accelerated by preconditioning the CG iteration, but we have not explored that topic in this paper. The limited memory BFGS method benefits from the fact that the stochastic Hessian information complements the curvature information provided by quasi-Newton updating.

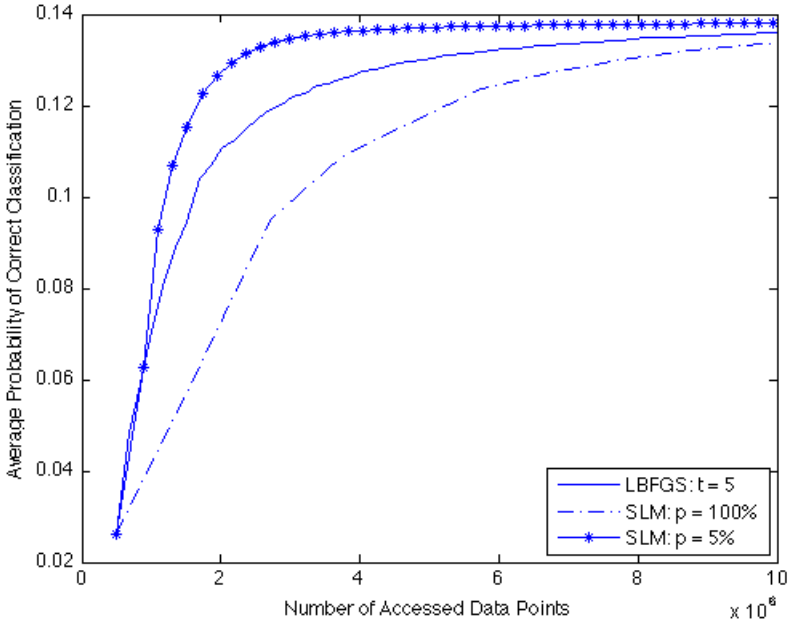

Figure 4: Comparison of L-BFGS vs. SLM with $p = 5\%, 100\%$, and $\max_{cg} = 5$

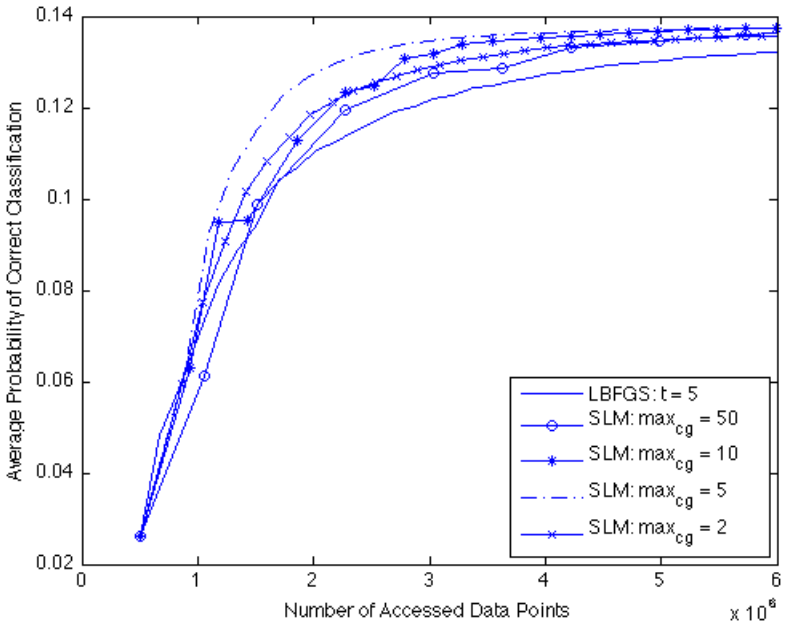

Figure 5: Behavior of Algorithm SLM, with $p = 5\%$, for varying $\max_{cg}$ values

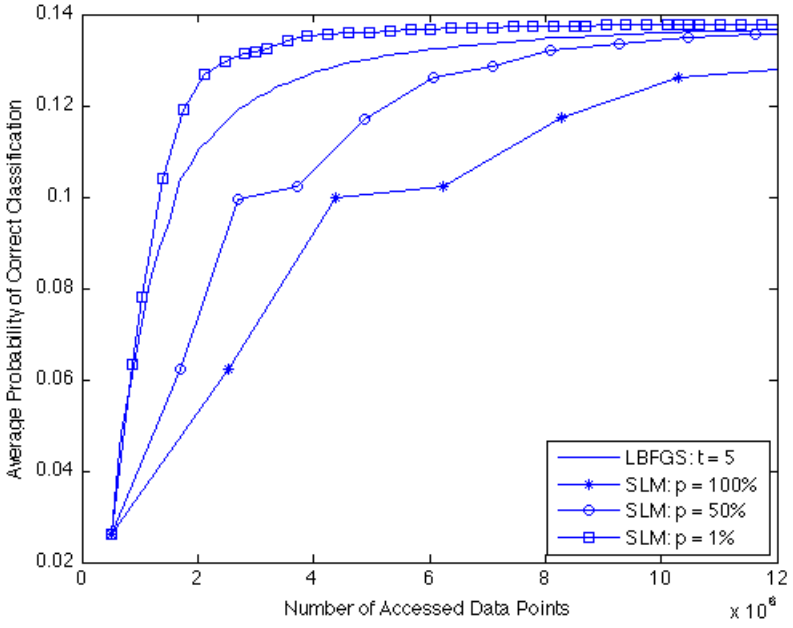

Figure 6: Behavior of Algorithm SLM, with $\max_{cg} = 10$, for $p = 1\%, 50\%$ and 100%

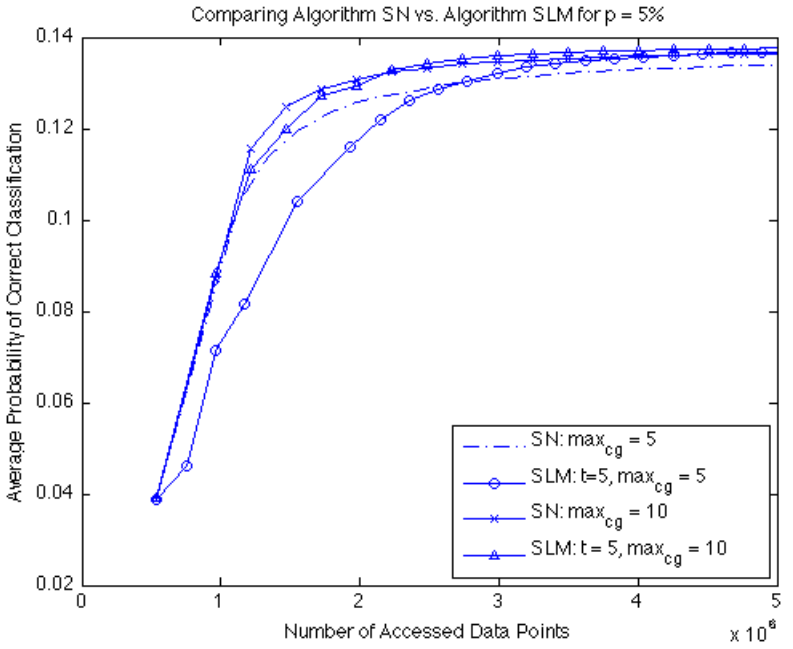

Figure 7: Comparing S-Newton vs SLM (with memory $t = 5$), for $\max_{cg} = 5, 10$.

This work was motivated, in part, by the availability of distributed computing environments that allow the parallel computation of very expensive loss functions involving many thousands of training points, as well as by the possibility of computing Hessian-vector products in parallel and at modest cost. Given the latency due to communication in such a setting, it is convenient to work with batch sizes that are not very small – and this in turn justifies the use of deterministic optimization techniques. The potential of the sub-sampled Hessian methods was illustrated on a speech recognition problem with data generated at Google.

Acknowledgments. The authors acknowledge several insightful conversations with Tito Homem-de-Mello, Alexander Shapiro and Yoram Singer.

References

[1] L. Bottou. *Online Learning and Neural Networks*, volume 17, chapter Online Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, 1998.

[2] R. Fletcher. *Practical Methods of Optimization*. J. Wiley and Sons, Chichester, England, second edition, 1987.

[3] I. Griva, S. G. Nash, and A. Sofer. *Linear and Nonlinear Optimization*. SIAM, Philadelphia, USA, second edition, 2008.

[4] S. Gunter J. Yu, S.V.N Vishwanathan and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. *The Journal of Machine Learning Research*, 11:1145–1200, 2010.

[5] C. Lin, R.C. Weng, and S. S. Keerthi. Trust region Newton method for logistic regression. *The Journal of Machine Learning Research*, 9:627–650, 2008.

[6] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[7] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, second edition, 1984.

[8] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning. Taipei, Taiwan*, 2002.

[9] T. P. Minka. A comparison of numerical optimizers for logistic regression. Technical report, Microsoft Research, 2003.

[10] S. G. Nash. Newton-type minimization via the Lanczos method. *SIAM Journal on Numerical Analysis*, 21(4), 1984.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, second edition, 2006.

[12] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407, 1951.

[13] A. Ruszczynski and A. Shapiro. *Stochastic Programming, Handbook in Operations Research and Management Science*. Elsevier Science, 2003.

[14] F. Sha and F. Pereira. Shallow parsing with conditional random fields. Technical report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA, USA, 2003.

[15] A. Shapiro and T. Homem de Mello. A simulation-based approach to two-stage stochastic programming with recourse. *Mathematical Programming*, 81:301–325, 1998.

[16] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[17] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88, London, 1981. Academic Press.