Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

A fast quasi-Newton-type method for large-scale stochastic optimisation ⋆

Adrian Wills * **Thomas B. Schön** ** **Carl Jidling** **

* *School of Engineering, University of Newcastle, Australia, (e-mail: adrian.wills@newcastle.edu.au).*
** *Department of Information Technology, Uppsala University, Sweden, (e-mail: {thomas.schon, carl.jidling}@it.uu.se)*

Abstract: In recent years there has been an increased interest in stochastic adaptations of limited memory quasi-Newton methods, which compared to pure gradient-based routines can improve the convergence by incorporating second-order information. In this work we propose a direct least-squares approach conceptually similar to the limited memory quasi-Newton methods, but that computes the search direction in a slightly different way. This is achieved in a fast and numerically robust manner by maintaining a Cholesky factor of low dimension. The performance is demonstrated on real-world benchmark problems which shows improved results in comparison with already established methods.

Keywords: Optimisation problems, Large-scale problems, Stochastic systems, Cholesky factorisation, Neural networks

1. INTRODUCTION

In learning algorithms we often face the classical and hard problem of stochastic optimisation where we need to minimise some non-convex cost function $f(x)$

$$\min_{x \in \mathbb{R}^d} f(x), \tag{1}$$

when we only have access to *noisy* evaluations of the cost function and its gradients. We take a particular interest in situations where the number of data and/or the number of unknowns d is very large.

The importance of this problem has been increasing for quite some time now. The reason is simple: many important applied problems ask for its solution, including most of the supervised machine learning algorithms when applied to large-scale settings. There are two important situations where the non-convex stochastic optimisation problem arise. Firstly, for large-scale problems it is often prohibitive to evaluate the cost function and its gradient on the entire dataset. Instead, it is divided into several mini-batches via subsampling, making the problem stochastic. This situation arise in most applications of deep learning. Secondly, when randomised algorithms are used to approximately compute the cost function and its gradients the result is always stochastic. This occurs for example in nonlinear system identification using the maximum likelihood method when particle filters are used to compute the intractable cost function and its gradients.

Our *contributions* are: 1. A new and efficient way of incorporating second-order (curvature) information into the stochastic optimiser via a *direct least-squares approach* conceptually similar to the popular limited memory quasi-Newton methods. 2. To facilitate a *fast and numerically robust implementation* we have derived tailored updating of a small dimension Cholesky factor given the new measurement pair (with dimension equal to the memory length). 3. The performance is also demonstrated on *real-world benchmark problems* which shows improved convergence properties over current state-of-the-art methods.

⋆ This research was financially supported by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012) and by the Swedish Research Council via the project *Learning flexible models for nonlinear dynamics* (contract number: 2017-03807).

2. RELATED WORK

Due to its importance, the stochastic optimisation problem is rather well studied by now. The first stochastic optimisation algorithm was introduced by Robbins and Monro (1951). It makes use of first-order information only, motivating the name stochastic gradient (SG), which is the contemporary term (Bottou et al., 2018) for these algorithms, originally referred to as stochastic approximation. Interestingly most SG algorithms are not descent methods since the stochastic nature of the update can easily produce a new iterate corresponding to an increase in the cost function. Instead, they are Markov chain methods in that their update rule defines a Markov chain.

The basic first-order SG algorithms have recently been significantly improved by the introduction of various noise reduction techniques, see e.g. (Johnson and Zhang, 2013; Schmidt et al., 2013; Konečný and Richtárik, 2017; Defazio et al., 2014).

The well-known drawback of all first-order methods is the lack of curvature information. Analogously to the deterministic setting, there is a lot to be gained in extracting and using second-order information that is maintained in the form of the Hessian matrix. The standard quasi-Newton method is the BFGS method, named after its inventors (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). In its basic form, this algorithm does not scale to the large-scale settings we are interested in. The idea of only making use of the most recent iterates and gradients in forming the inverse Hessian approximation was suggested by Nocedal (1980) and Liu and Nocedal (1989). The resulting L-BFGS method is computationally cheaper with a significantly reduced memory footprint. Due to its simplicity and good performance, this has become one of

the most commonly used second-order methods for large-scale problems. Our developments makes use of the same trick underlying L-BFGS, but it is carefully tailored to the stochastic setting.

Over the past decade we have witnessed increasing capabilities of these so-called *stochastic quasi-Newton methods*, the category to which our developments belong. The work by Schraudolph et al. (2007) developed modifications of BFGS and its limited memory version. There has also been a series of papers approximating the inverse Hessian with a diagonal matrix, see e.g. Bordes et al. (2009) and Duchi et al. (2011). The idea of exploiting regularisation together with BFGS was successfully introduced by Mokhtari and Ribeiro (2014). Later Mokhtari and Ribeiro (2015) also developed a stochastic L-BFGS algorithm without regularisation. The idea of replacing the stochastic gradient difference in the BFGS update with a subsampled Hessian-vector product was recently introduced by Byrd et al. (2016), and Wang et al. (2017) derived a damped L-BFGS method.

Over the past five years we have also seen quite a lot of fruitful activity in combining the stochastic quasi-Newton algorithms with various first-order noise reduction methods (Moritz et al., 2016; Gower et al., 2016). A thorough and forward-looking overview of SG and its use within a modern machine learning context is provided by Bottou et al. (2018). It also includes interesting accounts of possible improvements along the lines of first-order noise reduction and second-order methods.

3. COMPUTING THE QUASI-NEWTON SEARCH DIRECTION

A quasi-Newton method consists of two steps. In the first step, we compute a search direction p_k that specifies in which *direction* of the input space we should move at iteration k. The second step computes a step length α_k that determines how *far* we should move in this direction. We then obtain the next iterate as

$$x_{k+1} = x_k + \alpha_k p_k. \tag{2}$$

In this work we focus on the first step, and compute the step length using the line search described by Wills and Schön (2019).

We address the problem of computing a search direction based on having a limited memory available for storing previous gradients and associated iterates. The approach we adopt is similar to limited memory quasi-Newton methods, but here we employ a *direct least-squares estimate* of the inverse Hessian matrix rather than more well-known methods such as damped L-BFGS and L-SR1. In contrast to traditional quasi-Newton methods, this approach does not strictly require the Hessian matrix to be symmetric, not does it require the secant condition to be exactly fulfilled.

It may appear peculiar to relax these requirements. However, in this setting it is not obvious that enforced symmetry necessarily produces a better search direction. Furthermore, the secant condition relies upon the rather strong approximation that the Hessian matrix is constant between two subsequent iterations. Treating the condition less strictly might be helpful when that approximation is poor, perhaps especially in a stochastic environment. We construct a limited-memory inverse Hessian approximation in Section 3.1 and show how to update this representation in Section 3.2. Section 3.3 provides a means to ensure that a descent direction is calculated and Section 3.4 summarises the final algorithm.

3.1 Quasi-Newton Inverse Hessian Approximations

According to the secant condition (see e.g. Fletcher (1987)), the inverse Hessian matrix H_k should satisfy

$$H_k y_k = s_k, \tag{3}$$

where $s_k = x_k - x_{k-1}$ and $y_k = g_k - g_{k-1}$ with the gradient $g_k \triangleq \nabla f(x)|_{x=x_k}$. Since there are generally more unknown values in H_k than can be determined from y_k and s_k alone, quasi-Newton methods update H_k from a previous estimate H_{k-1} by solving regularised problems of the type

$$\begin{aligned} H_k = \arg\min_H \quad & \|H - H_{k-1}\|_{F,W}^2 \\ \text{s.t.} \quad & H = H^\mathsf{T}, \quad H y_k = s_k, \end{aligned} \tag{4}$$

where $\|X\|_{F,W}^2 = \|XW\|_F^2 = \operatorname{trace}(W^\mathsf{T} X^\mathsf{T} X W)$ and the choice of weighting matrix W results in different algorithms (see Hennig (2015) for an interesting perspective on this).

We employ a similar approach and determine H_k as the solution to the following regularised least-squares problem

$$H_k = \arg\min_H \|H Y_k - S_k\|_F^2 + \lambda \|H - \bar{H}_k\|_F^2, \tag{5}$$

where Y_k and S_k hold a limited number of past $\widehat{y}_k$'s and s_k's according to

$$Y_k \triangleq [\widehat{y}_{k-m+1}, \ldots, \widehat{y}_k], \qquad S_k \triangleq [s_{k-m+1}, \ldots, s_k]. \tag{6}$$

Here, $m << d$ is the memory limit and $\widehat{y}_k = \widehat{g}_k - \widehat{g}_{k-1}$ where $\widehat{g}_k$ is an estimate of g_k. The regulator matrix $\bar{H}_k$ acts as a prior on H and can be modified at each iteration k; for computational efficiency we choose it as a diagonal matrix, as discussed in Section 3.4. The parameter $\lambda > 0$ is used to control the relative cost of the two terms in equation 5. It can be verified that the solution to the above least-squares problem (equation 5) is given by

$$H_k = \left(\lambda \bar{H}_k + S_k Y_k^\mathsf{T}\right) \left(\lambda I + Y_k Y_k^\mathsf{T}\right)^{-1}, \tag{7}$$

where I denotes the identity matrix. The above inverse Hessian estimate can be used to generate a search direction in the standard manner by scaling the negative gradient, that is

$$\widehat{p}_k = -H_k \widehat{g}_k. \tag{8}$$

We use $\widehat{p}_k$ to distinguish the search direction computed in the stochastic setting from its deterministic counterpart. For large-scale problems, the computation in equation 8 is not practical since it involves the inverse of a large matrix. To ameliorate this difficulty, we adopt the standard approach by storing only a minimal (limited memory) representation of the inverse Hessian estimate H_k. To describe this, note that the dimensions of the matrices involved are

$$H_k \in \mathbb{R}^{d \times d}, \qquad Y_k \in \mathbb{R}^{d \times m}, \qquad S_k \in \mathbb{R}^{d \times m}. \tag{9}$$

We can employ the Sherman–Morrison–Woodbury formula to arrive at the following equivalent expression for H_k

$$H_k = \left(\bar{H}_k + \lambda^{-1} S_k Y_k^\mathsf{T}\right) \left[I - Y_k \left(\lambda I + Y_k^\mathsf{T} Y_k\right)^{-1} Y_k^\mathsf{T}\right].$$

Importantly, the matrix inverse $\left(\lambda I + Y_k^\mathsf{T} Y_k\right)^{-1}$ is now by construction a positive definite matrix of size $m \times m$. Therefore, we construct and maintain a Cholesky factor of $\lambda I + Y_k^\mathsf{T} Y_k$ since this leads to efficient computations. In particular, if we express this matrix via a Cholesky decomposition

$$R_k^\mathsf{T} R_k = \lambda I + Y_k^\mathsf{T} Y_k, \tag{10}$$

where $R_k \in \mathbb{R}^{m\times m}$ is an upper triangular matrix, then the search direction $\widehat{p}_k = -H_k\widehat{g}_k$ can be computed via

$$\widehat{p}_k = -\bar{H}_k z_k - \lambda^{-1}S_k(Y_k^\mathsf{T} z_k), \tag{11a}$$
$$z_k = \widehat{g}_k - Y_k w_k, \tag{11b}$$
$$w_k = R_k^{-1}\left(R_k^{-\mathsf{T}}\left(Y_k^\mathsf{T}\widehat{g}_k\right)\right). \tag{11c}$$

Note that the above expressions for $\widehat{p}_k$ involve first computing w_k, which itself involves a computationally efficient forward-backward substitution (recalling that R_k is an $m \times m$ upper triangular matrix and the memory length m is typically 10–50). Furthermore, $\bar{H}_k$ is typically diagonal so that $\bar{H}_k z_k$ is also efficient to compute. The remaining operations involve four matrix-vector products and two vector additions. Therefore, for problems where $d >> m$ the matrix-vector products will dominate the computational cost.

Constructing R_k can be achieved in several ways. The so-called normal-equation method constructs the (upper triangular) part of $\lambda I + Y_k^\mathsf{T} Y_k$ and then employs a Cholesky routine, which produces R_k in $O(n\frac{m(m+1)}{2} + m^3/3)$ operations. Alternatively, we can compute R_k by applying Givens rotations or Householder reflections to the matrix $M_k = \left[\sqrt{\lambda}I \; Y_k^\mathsf{T}\right]^\mathsf{T}$. This costs $O(2m^2((n+m) - m/3))$ operations, and is therefore more expensive, but typically offers better numerical accuracy (Golub and Van Loan, 2012).

3.2 Fast and robust inclusion of new measurements

In order to maximise the speed, we have developed a method for updating a Cholesky factor given the new measurement pair $(s_{k+1}, \widehat{y}_{k+1})$. Suppose we start with a Cholesky factor R_k at iteration k such that

$$R_k^\mathsf{T} R_k = \lambda I + Y_k^\mathsf{T} Y_k. \tag{12}$$

Assume, without loss of generality, that Y_k and S_k are ordered in the following manner

$$Y_k \triangleq [\mathcal{Y}_1, \widehat{y}_{k-m+1}, \mathcal{Y}_2], \quad S_k \triangleq [\mathcal{S}_1, s_{k-m+1}, \mathcal{S}_2], \tag{13}$$

where $\mathcal{Y}_1$, $\mathcal{Y}_2$, $\mathcal{S}_1$ and $\mathcal{S}_2$ are defined as

$$\mathcal{Y}_1 \triangleq [\widehat{y}_{k-m+\ell+1}, \ldots, \widehat{y}_k], \quad \mathcal{Y}_2 \triangleq [\widehat{y}_{k-m+2}, \ldots, \widehat{y}_{k-m+\ell}], \tag{14a}$$
$$\mathcal{S}_1 \triangleq [s_{k-m+\ell+1}, \ldots, s_k], \quad \mathcal{S}_2 \triangleq [s_{k-m+2}, \ldots, s_{k-m+\ell}], \tag{14b}$$

and ℓ is an appropriate integer so that Y_k and S_k have m columns. The above ordering arises from "wrapping-around" the index when storing the measurements. We create the new Y_{k+1} and S_{k+1} by replacing the oldest column entries, $\widehat{y}_{k-m+1}$ and s_{k-m+1}, with the latest measurements $\widehat{y}_{k+1}$ and s_{k+1}, respectively, so that

$$Y_{k+1} \triangleq [\mathcal{Y}_1, \widehat{y}_{k+1}, \mathcal{Y}_2], \qquad S_{k+1} \triangleq [\mathcal{S}_1, s_{k+1}, \mathcal{S}_2]. \tag{15}$$

The aim is to generate a new Cholesky factor R_{k+1} such that

$$R_{k+1}^\mathsf{T} R_{k+1} = \lambda I + Y_{k+1}^\mathsf{T} Y_{k+1}. \tag{16}$$

To this end, let the upper triangular matrix R_k be written conformally with the columns of Y_k as

$$R_k = \begin{bmatrix} \mathcal{R}_1 & r_1 & \mathcal{R}_2 \\ \cdot & r_2 & r_3 \\ \cdot & \cdot & \mathcal{R}_4 \end{bmatrix}, \tag{17}$$

so that $\mathcal{R}_1$ and $\mathcal{R}_2$ have the same number of columns as $\mathcal{Y}_1$ and $\mathcal{Y}_2$, respectively. Furthermore, r_1 is a column vector, r_2 is a scalar and r_3 is a row vector. Therefore,

$$\begin{aligned} R_k^\mathsf{T} R_k &= \begin{bmatrix} \mathcal{R}_1^\mathsf{T}\mathcal{R}_1 & \mathcal{R}_1^\mathsf{T} r_1 & \mathcal{R}_1^\mathsf{T}\mathcal{R}_2 \\ \cdot & r_2^2 + r_1^\mathsf{T} r_1 & r_1^\mathsf{T}\mathcal{R}_2 + r_2 r_3 \\ \cdot & \cdot & \mathcal{R}_4^\mathsf{T}\mathcal{R}_4 + \mathcal{R}_2^\mathsf{T}\mathcal{R}_2 + r_3^\mathsf{T} r_3 \end{bmatrix} \\ &= \begin{bmatrix} \lambda I + \mathcal{Y}_1^\mathsf{T}\mathcal{Y}_1 & \mathcal{Y}_1^\mathsf{T}\widehat{y}_{k-m+1} & \mathcal{Y}_1^\mathsf{T}\mathcal{Y}_2 \\ \cdot & \lambda + \widehat{y}_{k-m+1}^\mathsf{T}\widehat{y}_{k-m+1} & \widehat{y}_{k-m+1}^\mathsf{T}\mathcal{Y}_2 \\ \cdot & \cdot & \lambda I + \mathcal{Y}_2^\mathsf{T}\mathcal{Y}_2 \end{bmatrix}. \end{aligned} \tag{18}$$

By observing a common structure for the update $\lambda I + Y_{k+1}^\mathsf{T} Y_{k+1}$ it is possible to write

$$\begin{aligned} \lambda I + Y_{k+1}^\mathsf{T} Y_{k+1} &= \\ &= \begin{bmatrix} \lambda I + \mathcal{Y}_1^\mathsf{T}\mathcal{Y}_1 & \mathcal{Y}_1^\mathsf{T}\widehat{y}_{k+1} & \mathcal{Y}_1^\mathsf{T}\mathcal{Y}_2 \\ \cdot & \lambda + \widehat{y}_{k+1}^\mathsf{T}\widehat{y}_{k-m+1} & \widehat{y}_{k+1}^\mathsf{T}\mathcal{Y}_2 \\ \cdot & \cdot & \lambda I + \mathcal{Y}_2^\mathsf{T}\mathcal{Y}_2 \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{R}_1^\mathsf{T}\mathcal{R}_1 & \mathcal{R}_1^\mathsf{T} r_4 & \mathcal{R}_1^\mathsf{T}\mathcal{R}_2 \\ \cdot & r_5^2 + r_4^\mathsf{T} r_4 & r_4^\mathsf{T}\mathcal{R}_2 + r_5 r_6 \\ \cdot & \cdot & \mathcal{R}_6^\mathsf{T}\mathcal{R}_6 + \mathcal{R}_2^\mathsf{T}\mathcal{R}_2 + r_6^\mathsf{T} r_6 \end{bmatrix}, \end{aligned} \tag{19}$$

where r_4, r_5 and r_6 are determined by

$$r_4 = \mathcal{R}_1^{-\mathsf{T}}(\mathcal{Y}_1^\mathsf{T}\widehat{y}_{k+1}), \tag{20a}$$
$$r_5 = \left(\lambda + \widehat{y}_{k+1}^\mathsf{T}\widehat{y}_{k+1} - r_4^\mathsf{T} r_4\right)^{1/2}, \tag{20b}$$
$$r_6 = \frac{1}{r_5}\left(\widehat{y}_{k+1}^\mathsf{T}\mathcal{Y}_2 - r_4^\mathsf{T}\mathcal{R}_2\right). \tag{20c}$$

The final term $\mathcal{R}_6$ can be obtained by noticing that

$$\mathcal{R}_6^\mathsf{T}\mathcal{R}_6 + \mathcal{R}_2^\mathsf{T}\mathcal{R}_2 + r_6^\mathsf{T} r_6 = \mathcal{R}_4^\mathsf{T}\mathcal{R}_4 + \mathcal{R}_2^\mathsf{T}\mathcal{R}_2 + r_3^\mathsf{T} r_3, \tag{21}$$

which implies

$$\mathcal{R}_6^\mathsf{T}\mathcal{R}_6 = \mathcal{R}_4^\mathsf{T}\mathcal{R}_4 - r_6^\mathsf{T} r_6 + r_3^\mathsf{T} r_3. \tag{22}$$

Therefore $\mathcal{R}_6$ can be obtained in a computationally very efficient manner by down-dating and updating the Cholesky factor $\mathcal{R}_4$ with the rank-1 matrices $r_6^\mathsf{T} r_6$ and $r_3^\mathsf{T} r_3$, respectively (see e.g. Section 12.5.3 in Golub and Van Loan (2012)).

3.3 Ensuring a descent direction

In deterministic quasi-Newton methods, the search direction p_k must be chosen to ensure a descent direction such that $p_k^\mathsf{T} g_k < 0$, since this guarantees reduction in the cost function for sufficiently small step lengths α_k. Since $p_k = -Hg_k$, we have that $p_k^\mathsf{T} g_k = -g_k^\mathsf{T} H_k g_k$ which is always negative if the approximation H_k of the inverse Hessian is positive definite. Otherwise, we can modify the search direction by subtracting a multiple of the gradient $p_k \leftarrow p_k - \beta_k g_k$. This is motivated by noticing that

$$(p_k - \beta g_k)^\mathsf{T} g_k = p_k^\mathsf{T} g_k - \beta_k g_k^\mathsf{T} g_k, \tag{23}$$

which always can be made negative by selecting β_k large enough, i.e. if

$$\beta_k > \frac{p_k^\mathsf{T} g_k}{g_k^\mathsf{T} g_k}. \tag{24}$$

In the stochastic setting, the condition above does not strictly enforce a descent direction. Hence, the search direction $\widehat{p}_k$ as determined by equation 8 is not a descent direction in general. Nevertheless, picking β_k sufficiently large ensures that

$$\mathrm{E}\left[(\widehat{p}_k - \beta_k\widehat{g}_k)^\mathsf{T}\widehat{g}_k\right] < 0, \tag{25}$$

i.e. the condition is fulfilled in expectation. In practise, we use a heuristic approach where the deterministic quantities in equation 24 are replaced by their stochastic counterparts. We should perhaps stress that the need for this modification occurred very infrequently in the experiments.

3.4 Resulting algorithm

We summarise the ideas from this section in Algorithm 1. The if-statement on line 5 is included to provide a safeguard against numerical instability. Note that the method relies on a user supplied $\bar{H}_k$, a choice that is by no means obvious. However, in practise it has proven very useful to employ a simple strategy of choosing $\bar{H}_k \triangleq \gamma_k I$, where the positive scalar $\gamma_k > 0$ is adaptively chosen in each iteration. Recall that α_k is the step length at iteration k, then as a crude measure of progress we adopt the following rule

$$\gamma_k = \begin{cases} \kappa\gamma_{k-1}, & \text{if } \alpha_{k-1} = 1, \\ \gamma_{k-1}/\kappa, & \text{if } \alpha_{k-1} < 1/\rho^q, \\ \gamma_{k-1}, & \text{otherwise,} \end{cases} \tag{26}$$

where $\rho \in (0, 1)$ is a scale factor used in the line search. Moreover, $\kappa \geq 1$ is a scale parameter, and q corresponds to the number of backtracking loops in the line search; the values $\kappa = 1.3$ and $q = 3$ were found to work well in practise. The intuition behind equation 26 is that if no modification of the step length α_k is made, we can allow for a more "aggressive" regularisation. Note that a low γ_k is favouring small elements in H_k. Since $\widehat{p}_k = -H_k\widehat{g}_k$, this limits the magnitude of $\widehat{p}_k$ and the change $\|x_{k+1} - x_k\|$ is kept small. Hence, it is good practice to set the initial scaling γ_0 relatively small and then let it scale up as the optimisation progresses. Furthermore, we should point out that a diagonal $\bar{H}_k$ comes with an efficiency benefit, since the product $\bar{H}_k z_k$ in equation 11a then is obtained as the element-wise product between two vectors.

Algorithm 1 Limited memory least-squares (LMLS)

Require: An initial estimate x_1, a maximum number of iterations $k_{\max}$, memory limit m, regularisation parameter λ, scale factor $\rho \in (0, 1)$.

1: Set $k = 1$
2: **while** $k < k_{\max}$ **do**
3: Obtain a measurement of the cost function and its gradient

$$\widehat{f}_k = f(x_k) + e_k, \qquad \widehat{g}_k = g_k + v_k.$$

4: Set $\bar{H}_k = \gamma_k I$ where

$$\gamma_k = \begin{cases} \kappa\gamma_{k-1}, & \text{if } \alpha_{k-1} = 1, \\ \gamma_{k-1}/\kappa, & \text{if } \alpha_{k-1} < 1/\rho^q, \\ \gamma_{k-1}, & \text{otherwise.} \end{cases}$$

5: **if** $\widehat{y}_k^{\mathsf{T}} s_k > \epsilon \|s_k\|_2^2$ **then**
6: **if** $k > m$ **then**
7: Replace oldest vector in Y_{k-1} with $\widehat{y}_k$ to form Y_k.
8: Replace oldest vector S_{k-1} with s_k to form S_k.
9: **else**
10: Form Y_k by adding $\widehat{y}_k$ to Y_{k-1}.
11: Form S_k by adding s_k to S_{k-1}.
12: **end if**
13: **else**
14: Set $Y_k = Y_{k-1}$ and $S_k = S_{k-1}$
15: **end if**
16: Select $\widehat{p}_k$ as (see Section 3.2)

$$\begin{aligned} \widehat{p}_k &= -\bar{H}_k z_k - \lambda^{-1} S_k (Y_k^{\mathsf{T}} z_k), \\ z_k &= \widehat{g}_k - Y_k w_k, \\ w_k &= R_k^{-1}\left(R_k^{-\mathsf{T}}\left(Y_k^{\mathsf{T}}\widehat{g}_k\right)\right). \end{aligned}$$

17: **if** $\widehat{p}_k^{\mathsf{T}}\widehat{g}_k > 0$ **then**
18: Set $\widehat{p}_k \leftarrow \widehat{p}_k - \beta_k\widehat{g}_k$ where $\beta_k > \frac{\widehat{p}_k^{\mathsf{T}}\widehat{g}_k}{\widehat{g}_k^{\mathsf{T}}\widehat{g}_k}$.
19: **end if**
20: Select α_k via line search by Wills and Schön (2019).
21: Update $x_{k+1} = x_k + \alpha_k\widehat{p}_k$
22: Set $k \leftarrow k + 1$
23: **end while**

4. NUMERICAL EXPERIMENTS

Let us now put our new developments to the test on a suite of problems from different categories to exhibit different properties and challenges. In Section 4.1 we study a commonly used benchmark, namely the collection of logistic classification problems described by Chang and Lin (2011) in the form of their library for support vector machines (LIBSVM). In Section 4.2 we consider an optimisation problem arising from the use of deep learning to solve the classical machine learning benchmark MNIST [1], where the task is to classify images of handwritten digits. Also, we test our method on training a neural network on the CIFAR-10 dataset (Krizhevsky, 2009). Finally, in Section 4.3 we consider parameter optimisation in a nonlinear state space model.

In our experiments we compare against relevant state-of-the-art methods. All experiments were run on a MacBook Pro 2.8GHz laptop with 16GB of RAM using Matlab 2018b. All routines where programmed in C and compiled via Matlab's `mex` command and linked against Matlab's Level-1,2 BLAS libraries. The chosen algorithm parameters are given in Table 1.

4.1 Logistic loss and a 2-norm regulariser

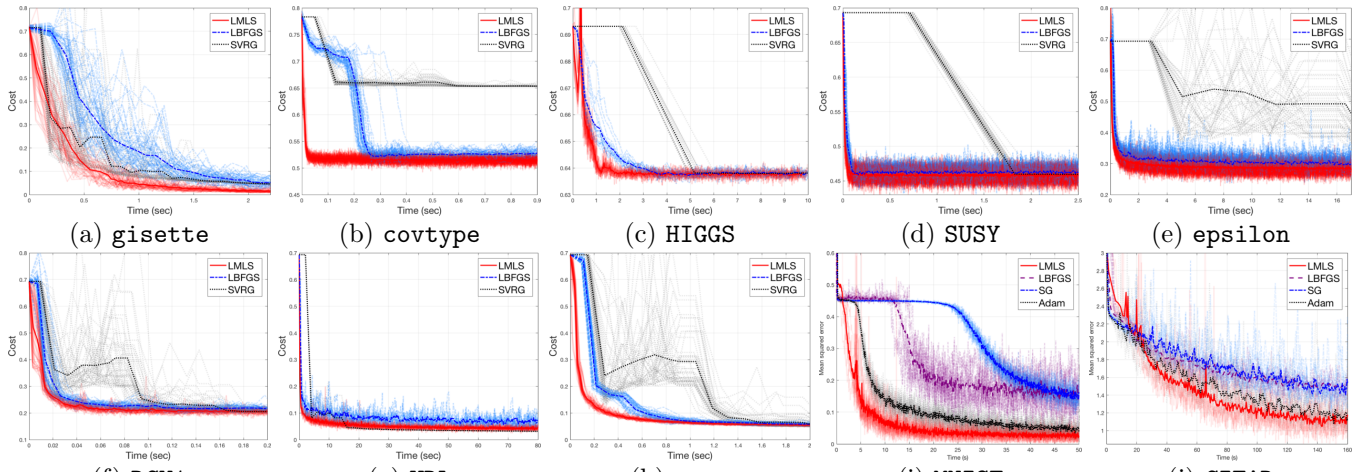
The task here is to solve eight different empirical risk minimisation problems using a logistic loss function with an L2 regulariser. The data is taken from Chang and Lin (2011). These problems are commonly used for profiling optimisation algorithms of the kind introduced in this paper, facilitating comparison with existing state-of-the-art algorithms. More specifically, we have used a similar set-up as Gower et al. (2016), which inspired this study.

We compared our limited memory least-squares approach (denoted as `LMLS`) against two existing methods from the literature, namely, the limited memory stochastic BFGS method after Bollapragada et al. (2018) (denoted as `LBFGS`) and the stochastic variance reduced gradient descent (SVRG) by Johnson and Zhang (2013) (denoted `SVRG`). Figures 1a to 1h show the cost versus time for 50 Monte-Carlo experiments. Our `LMLS` method demonstrates a fast convergence and outperforms the other methods across all the eight problems.

4.2 Deep learning

Deep convolutional neural networks (CNNs) with multiple layers of convolution, pooling and nonlinear activation functions are delivering state-of-the-art results on many tasks in computer vision. We are here borrowing the stochastic optimisation problems arising in using such a deep CNN to solve the MNIST and CIFAR-10 benchmarks. The particular CNN structure used for the MNIST example employs 5×5 convolution kernels, pooling layers and a fully connected layer at the end. We made use of the publicly available code provided by Zhang (2016), which contains all the implementation details. For the CIFAR-10 example, the network includes 13 layers with more than 150,000 weights. The MATLAB toolbox *MatConvNet* (Vedaldi and Lenc, 2015) was used in the implementation. Figures 1i and 1j show the average cost versus time for 10 Monte-Carlo trials with four different algorithms: 1. the method developed here (`LMLS`), 2. a stochastic limited memory BFGS method after Bollapragada et al. (2018) (denoted `LBFGS`), 3. `Adam` developed by Kingma and Ba

[1] `yann.lecun.com/exdb/mnist/`



(a) gisette (b) covtype (c) HIGGS (d) SUSY (e) epsilon

(f) RCV1 (g) URL (h) spam (i) MNIST (j) CIFAR

Fig. 1. Performance on two classification tasks using a logistic loss with a two-norm regulariser ((a)-(h)), and two deep convolutional neural networks (CNNs) used for recognising images of handwritten digits from the MNIST data (i), and classification of the images in the CIFAR-10 data (j). Lighter shaded lines indicate individual runs, whereas the darker shaded line indicates the average.

(2015), and 4. stochastic gradient (denoted SG). Note that all algorithms make use of the same gradient code. Also for these problems, our LMLS method performs well – apart from in the initial phases, it converges faster than the remaining methods.

Table 1. Datasets used in experiments; n: dataset size; d: number of variables; remaining columns list design parameters, including the mini-batch size b.

Dataset	n	d	b	m	λ	γ_0
gisette	6 000	5 000	770	20	10^{-4}	10
covtype	581 012	54	7 620	55	10^{-4}	100
HIGGS	11 000 000	28	66 340	28	10^{-4}	100
SUSY	3 548 466	18	5 000	18	10^{-4}	100
epsilon	400 000	2 000	1 000	20	10^{-4}	100
rcv1	20 242	47 236	710	10	10^{-4}	600
URL	2 396 130	3 231 961	1548	5	10^{-4}	100
spam	82 970	823 470	2048	2	10^{-4}	200
MNIST	60 000	3898	1000	50	$8 \cdot 10^{-4}$	1
CIFAR	50 000	150 000	200	20	10^{-2}	0.5

4.3 A nonlinear state-space model

In this subsection, we consider Algorithm 1 (LMLS) as applied to maximum-likelihood parameter estimation for a nonlinear state-space model (see e.g. Schön et al. (2011)). In particular, a commonly employed nonlinear benchmark problem involves the following system

$$x_{t+1} = ax_t + b\frac{x_t}{1+x_t^2} + c\cos(1.2t) + v_t, \tag{28a}$$

$$y_t = dx_t^2 + e_t, \tag{28b}$$

$$\begin{bmatrix} v_t \\ e_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q & 0 \\ 0 & r \end{bmatrix} \right), \tag{28c}$$

where the true parameters are $\theta^\star = [a^\star, b^\star, c^\star, d^\star, q^\star, r^\star] = [0.5, 25, 8, 0.05, 0, 0.1]$. We repeat the simulation experiment from Schön et al. (2011), where a Monte Carlo study was performed using 100 different data realisations $y_{1:N}$ of length $N = 100$. For each of these cases, an estimate $\widehat{\theta}$ was computed using 100 iterations of Algorithm 1. The algorithm was initialised with the i'th element of θ_0 chosen via $\theta_0(i) \sim \mathcal{U}\left(\frac{1}{2}\theta^\star(i), \frac{3}{2}\theta^\star(i)\right)$. In all cases $M = 50$ particles were used in order to compute the log-likelihood cost function and its associated gradient vector.

Figure 2 shows the parameter iterates (for a, b, c, d). This shows all Monte Carlo runs (note that none were trapped in a local minima). By way of comparison, the method presented in this paper is compared with the EM approach from Schön et al. (2011) (denoted PSEM). The results are provided in Table 2, where the values are the sample mean of the parameter estimate across the Monte Carlo trials plus/minus the sample standard deviation. For the EM approach, 8/100 simulations were trapped in minima that were far from the global minimum and these results have been removed from the calculations in Table 2.

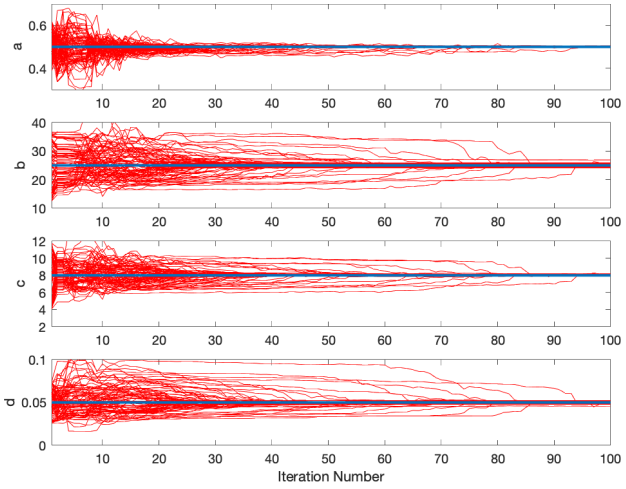


Fig. 2. Parameter iterations for nonlinear benchmark problem. True value (solid blue line), parameter evolution (one red line per simulation).

Table 2. True and estimated parameter values for LMLS and PSEM algorithms; mean value and standard deviations are shown for the estimates based on 100 Monte Carlo runs.

θ	$\theta^\star$	LMLS	PSEM
a	0.5	0.50 ± 0.0011	0.50 ± 0.0019
b	25.0	25.1 ± 0.43	25.0 ± 0.99
c	8.0	8.0 ± 0.06	7.99 ± 0.13
d	0.05	0.05 ± 0.001	0.05 ± 0.0026
q	0	$1 \times 10^{-4} \pm 6 \times 10^{-4}$	$7.78 \times 10^{-5} \pm 7.6 \times 10^{-5}$
r	0.1	0.1 ± 0.015	0.106 ± 0.015

5. CONCLUSION AND FUTURE WORK

In this work we have presented a least-squares based limited memory optimisation routine that benefits from second order information by approximating the inverse Hessian matrix. The procedure is conceptually similar to quasi-Newton methods, although we do not explicitly enforce symmetry or satisfaction of the secant condition. By regularising with respect to an inverse Hessian prior, we allow for an adaptive aggressiveness in the search direction update. We have shown that the computations can be made robust and efficient using tailored Cholesky decompositions, with a cost that scales linearly in the problem dimension. The method shows improved convergence properties over existing algorithms when applied to benchmark problems of various size and complexity.

REFERENCES

Bollapragada, R., Mudigere, D., Nocedal, J., Shi, H.J.M., and Tang, P.T.P. (2018). A progressive batching L-BFGS method for machine learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Stockholm, Sweden.

Bordes, A., Bottou, L., and Gallinari, P. (2009). SGDQN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research (JMLR)*, 10, 1737–1754.

Bottou, L., Curtis, F.E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2), 223–311.

Broyden, C.G. (1967). Quasi-Newton methods and their application to function minimization. *Mathematics of Computation*, 21, 368–381.

Byrd, R.H., Hansen, S.L., Nocedal, J., and Singer, Y. (2016). A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2), 1008–1031.

Chang, C.C. and Lin, C.J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 27:1–27:27.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: a fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems (NIPS)*. Montréal, Canada.

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

Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, 13(3), 317–322.

Fletcher, R. (1987). *Practical methods of optimization*. John Wiley & Sons, Chichester, UK, second edition.

Goldfarb, D. (1970). A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24(109), 23–26.

Golub, G.H. and Van Loan, C.F. (2012). *Matrix Computations*. John Hopkins University Press, Baltimore, fourth edition.

Gower, R.M., Goldfarb, D., and Richtarik, P. (2016). Stochastic block BFGS: squeezing more curvature out of data. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. New York, NY, USA.

Hennig, P. (2015). Probabilistic interpretation of linear solvers. *SIAM Journal on Optimization*, 25(1), 234–260.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NIPS)*. Lake Tahoe, NV, USA.

Kingma, D.P. and Ba, J. (2015). Adam: a method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations (ICLR)*. San Diego, CA, USA.

Konečný, J. and Richtárik, P. (2017). Semi-stochastic gradient descent methods. *Frontiers in Applied Mathematics and Statistics*, 3(9).

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Liu, D.C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3), 503–528.

Mokhtari, A. and Ribeiro, A. (2014). RES: regularized stochastic BFGS algorithm. *IEEE Transactions on Signal Processing*, 62(23), 6089–6104.

Mokhtari, A. and Ribeiro, A. (2015). Global convergence of online limited memory BFGS. *Journal of Machine Learning Research (JMLR)*, 16, 3151–3181.

Moritz, P., Nishihara, R., and Jordan, M.I. (2016). A linearly-convergent stochastic L-BFGS algorithm. In *The 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Cadiz, Spain.

Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151), 773–782.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3), 400–407.

Schmidt, M., Le Roux, N., and Bach, F. (2013). Minimizing finite sums with the stochastic average gradient. Technical Report arXiv:1309.2388, arXiv preprint.

Schön, T.B., Wills, A., and Ninness, B. (2011). System identification of nonlinear state-space models. *Automatica*, 47(1), 39–49.

Schraudolph, N.N., Yu, J., and Günter, S. (2007). A stochastic quasi-Newton method for online convex optimization. In *Proceedings of the 11th international conference on Artificial Intelligence and Statistics (AISTATS)*.

Shanno, D.F. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111), 647–656.

Vedaldi, A. and Lenc, K. (2015). Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*.

Wang, X., Ma, S., Goldfarb, D., and Liu, W. (2017). Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2), 927–956.

Wills, A. and Schön, T.B. (2019). Stochastic quasi-newton with line-search regularization. Technical report, arXiv:1909.01238.

Zhang, Z. (2016). Derivation of backpropagation in convolutional neural networks (CNN). `github.com/ZZUTK/An-Example-of-CNN-on-MNIST-dataset`.