SIAM J. OPTIM.
Vol. 32, No. 1, pp. 228–255
© 2022 Society for Industrial and Applied Mathematics

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

QNG: A QUASI-NATURAL GRADIENT METHOD FOR LARGE-SCALE STATISTICAL LEARNING*

XIAOYU HE†, ZIBIN ZHENG‡, YUREN ZHOU†, AND CHUAN CHEN†

Abstract. Natural gradient method provides a powerful paradigm for training statistical models and offers several appealing theoretic benefits. It constructs the Fisher information matrix to correct the ordinary gradients, and thus, the cost may become prohibitively expensive in the large-scale setting. This paper proposes a quasi-natural gradient method to address this computational issue. It employs a rank-one model to capture the second-order information from the underlying statistical manifold and to iteratively update the Fisher approximations. A three-loop procedure is designed to implement the updating formulas. This procedure resembles the classical two-loop procedure in the limited-memory BFGS method but saves half of the memory usage while it can be made faster. The resulting method retains the convergence rate advantages of existing stochastic optimization methods and has inherent ability in handling nonconvexity. Numerical studies conducted on several machine learning tasks demonstrate the reduction in convergence time and the robustness in tackling nonconvexity relative to stochastic gradient descent and the online limited-memory BFGS method.

Key words. natural gradient, large-scale stochastic optimization, statistical learning, Fisher information matrix

AMS subject classifications. 90C06, 90C53, 65K05

DOI. 10.1137/20M1376753

1. Introduction.

Many applications in machine learning consider solving the empirical risk minimization problem formulated as

$$\min_{\theta\in\mathbb{R}^n} F(\theta) = \frac{1}{N}\sum_{i=1}^{N} f(\theta;\xi_i), \tag{1.1}$$

where f is continuously differentiable and $\xi_1,\dots,\xi_N$ denote independent training samples [8, 35]. We assume that the learning task is supervised, so each sample ξ consists of a pair of input vector x and target output z, i.e., $\xi = (x, z)$. The function f in this setting typically takes the form

$$f(\theta;\xi) = \ell(h(\theta, x), z) + R(\theta), \tag{1.2}$$

where h is a prediction model parameterized by θ, ℓ is a loss function measuring the disagreement between the prediction $h(\theta, x)$ and the target z, and R is a regularization term. As in most real-world applications, both n and N are assumed to be very large, indicating that neither evaluating the batch gradients nor applying second-order approximations to the objective function is applicable.

*Received by the editors October 29, 2020; accepted for publication (in revised form) October 13, 2021; published electronically March 14, 2022.

https://doi.org/10.1137/20M1376753

Funding: This work was funded by the Key-Area Research and Development Program of Guangdong Province (2020B010165003), the National Natural Science Foundation of China (62006252, 61773410, 62176269, 11801595), and the Guangdong Basic and Applied Basic Research Foundation (2021A1515011840).

†School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, People's Republic of China (hexy_gz@outlook.com, zhouyuren@mail.sysu.edu.cn, chenchuan@mail.sysu.edu.cn).

‡Corresponding author. School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, People's Republic of China (zhzibin@mail.sysu.edu.cn).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

This article concerns a special case of (1.2) when the loss function ℓ has certain statistical explanations. One example is the negative log-likelihood loss given by

$$\ell(h(\theta, x), z) = -\log(p_\theta(z|x)), \tag{1.3}$$

where $p_\theta(z|x) = p(z|h(\theta, x))$ denotes the density corresponding to the probability of correctly predicting the target z with the model $h(\theta, x)$. Representative learning tasks using this loss function include logistic regression and softmax regression and often lead to a standard maximum likelihood learning which abounds in probability-based learning approaches [25]. The negative log-likelihood loss coincides with the Kullback–Leibler divergence of the prediction distribution $p_\theta(z|x)$ from the output distribution $p(z|x)$ [21]. Therefore, minimizing the empirical expectation of loss can be explained as locating a point on the statistical manifold.

Another example, widely adopted in nonlinear regression and neural networks [40], is the least squares loss:

$$\ell(h(\theta, x), z) = \|z - h(\theta, x)\|^2. \tag{1.4}$$

It can be cast in a probabilistic setting using the relation $\sum_{i=1}^N \ell(h(\theta, x_i), z_i) \propto -\log \prod_{i=1}^N \exp(-\|z_i - h(\theta, x_i)\|^2)$. That is, optimizing with the least squares loss is equivalent to fitting a univariate Gaussian model with fixed variance.

The loss functions listed above represent a rich family of stochastic optimization problems whose search region can be transformed from the space of the parameter θ into a statistical manifold implied from the prediction model h. This suggests that exploring the underlying manifold structure may be beneficial to the optimization.

One straightforward way to solve the aforementioned problems is to optimize the objective in (1.1) in the parameter space $\mathbb{R}^n$, without taking into account the underlying manifold structure. Stochastic gradient descent (SGD) [31] is a pioneering work in this line and remains the most popular first-order algorithm. SGD is computationally efficient and easy to implement, but may suffer from slow convergence when handling ill-conditioned problems. This issue can be overcome by incorporating second-order information such as local curvature to correct the descent directions. Stochastic second-order methods usually approximate the Hessian using the classical BFGS update [27] while further reducing the computational budget with the limited-memory scheme [26, 19]. Representative examples include the online limited-memory BFGS (oLBFGS) method [33, 24] and the stochastic quasi-Newton method [9]. Although having the same convergence rate, stochastic second-order methods often work better than SGD when coping with ill-conditioning [1]. The price to pay for this is the computing cost (such as increasing both the runtime and memory usage by a constant but rather significant factor) and the need for stronger assumptions (such as the boundedness of all stochastic Hessian estimates).

An alternative approach is to perform gradient descent in the space $\mathcal{H}$ of prediction models rather than the space of specific parameters. Amari [2] popularized this idea and proposed the natural gradient method, which employs iterations of the form

$$\theta_{t+1} = \theta_t - \eta_t B^{-1}(\theta_t) \nabla F(\theta_t), \tag{1.5}$$

where η_t is the step size and $B(\theta_t)$ is the metric tensor characterizing the geometry of the space $\mathcal{H}$. The term $B^{-1}(\theta)\nabla F(\theta)$ is called natural gradient since it defines the steepest ascent direction of p_θ in the space $\mathcal{H}$. In the setting that $\mathcal{H}$ is a family of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

densities $p_\theta(z|x)$ parameterized by θ, the matrix $B(\theta)$ is usually referred to as the Fisher information matrix (or simply "Fisher") and can be calculated as

$$B(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{p_\theta(z|x_i)}\left[\nabla \log p_\theta(z|x_i) \nabla \log p_\theta(z|x_i)^T\right], \tag{1.6}$$

where $\mathbb{E}_{p_\theta(z|x_i)}$ denotes the expectation with respect to the prediction distribution $p_\theta(z|x_i)$. The natural gradient method is usually considered as belonging to the class of second-order methods due to its close connection to Newton's method: the Fisher $B(\theta)$ approaches the Hessian of the negative log-likelihood when θ approaches the optimum [15]. However, these two matrices do not coincide in general, because the Hessian depends only on the data distribution while the Fisher also depends on the prediction distribution.[1] Apart from this difference, the natural gradient method possesses two additional advantages: (1) the Fisher is always positive semidefinite and (2) the descent direction is invariant to reparameterization of the model. This means the natural gradient method retains the ability to capture local curvature information while it can easily handle nonconvex objectives.

This work focuses on the natural gradient method in large-scale settings. As in SGD, one can easily reduce the cost of gradient evaluations through replacing $\nabla F(\theta_t)$ in (1.5) by a stochastic estimate. Calculating the inverse of the Fisher, $B^{-1}(\theta_t)$, is more difficult, because the Fisher is an $n \times n$ matrix averaged over all N inputs. This has resulted in the development of stochastic approximation methods for calculating the Fisher or its inverse [28, 4, 32, 22]. Existing works generally assume n is small, or require the minibatch size to be sufficiently small to facilitate an incremental update, or rely on problem-dependent diagonal approximation schemes, highlighting the need for new methods that can efficiently handle generic large-scale problems.

In the next section we present a quasi-natural gradient (QNG) method for handling large-scale problem (1.1) whose search space admits a statistical manifold structure. The QNG method resembles quasi-Newton methods in the sense that it updates an approximation of the inverse of the Fisher and avoids recomputing the Fisher or its inverse at every iteration. We propose a rank-one model for the stochastic Fisher estimates which works well in the minibatch setting when N is large. To reduce the algorithm complexity when n is large, we apply the limited-memory scheme and implement a three-loop procedure which is similar to but can be faster than the well-known two-loop procedure for the L-BFGS algorithm [19]. The simulation results using several machine learning tasks suggest that the proposed QNG method is robust and efficient.

The paper is organized as follows. The new algorithm is described in section 2 and its convergence properties are discussed in section 3. Experiments conducted on five machine learning tasks are reported in section 4. A literature survey on related methods is given in section 5. We conclude this paper and give some remarks in section 6.

Notation. For a vector v we use $\|v\|$ to denote its ℓ_2 norm. The notation A^{-T} refers to the inverse of the transpose of the matrix A. We use I for the identity matrix of appropriate dimension.

[1] The Hessian of a finite-sum log-likelihood optimization problem is $H(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla^2 \log p_\theta(z_i|x_i)$. Note its difference to the Fisher given in (1.6): in the Hessian z_i is directly taken from the ith training sample $\xi_i = (x_i, z_i)$, whereas the Fisher requires taking the expectation over $z \sim p_\theta(z|x_i)$. So evaluating the Fisher relies on both the distribution of training input x_i and the corresponding prediction model $p_\theta(z|x_i)$, but does not involve the training output z_i. For more details on their differences, please refer to [15, 21].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

2. Algorithm definition. To apply the natural gradient iterations (1.5) in large-scale settings, we use stochastic estimates of the gradient as well as the stochastic approximations to the Fisher. For a subset $\mathcal{S} \subset \{1, \dots, N\}$, we define the subsampled gradient and Fisher as

$$\nabla F(\theta; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f(\theta; \xi_i) \quad \text{and} \quad B(\theta; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} B(\theta; x_i), \tag{2.1}$$

where $B(\theta; x_i)$ denotes the instantaneous Fisher corresponding to the sample $\xi_i = (x_i, z_i)$:

$$B(\theta; x_i) = \mathbb{E}_{p_\theta(z|x_i)} \left[\nabla \log p_\theta(z|x_i) \nabla \log p_\theta(z|x_i)^T \right]. \tag{2.2}$$

Note that $B(\theta; x_i)$ does not depend on the target output z_i. The stochastic Fisher estimate $B(\theta; \mathcal{S})$ may have a maximum rank $\sum_{i=1}^{|\mathcal{S}|} \mathrm{Rank}(B(\theta; x_i))$; using it to approximate the Fisher requires $|\mathcal{S}|$ to be sufficiently large, which still could be prohibitively expensive.

We propose that an effective approach to further reduce the cost for representing the stochastic Fisher is to use an unbiased rank-one estimation. More precisely, we seek a vector $v(\theta; \mathcal{S}) \in \mathbb{R}^n$ such that

$$\mathbb{E}_{p_\theta(z|\mathcal{S})}[v(\theta; \mathcal{S}) v^T(\theta; \mathcal{S})] = B(\theta; \mathcal{S}), \tag{2.3}$$

where $\mathbb{E}_{p_\theta(z|\mathcal{S})}[\cdot]$ denotes the expectation taken with respect to all prediction models in $\mathcal{S}$. The vector $v(\theta; \mathcal{S})$ encodes the expected second-order information of the prediction model, playing a similar role as the stochastic Fisher but in a more compact form.

Our method for calculating $v(\theta; \mathcal{S})$ consists of two steps: (1) Sample for each $i \in \mathcal{S}$ an output $\hat{z}_i$ from the prediction distribution $p_\theta(z|x_i)$. (2) Compute $v(\theta; \mathcal{S})$ as

$$v(\theta; \mathcal{S}) = \frac{1}{\sqrt{|\mathcal{S}|}} \sum_{i \in \mathcal{S}} \nabla \log p_\theta(\hat{z}_i|x_i), \quad \hat{z}_i \sim p_\theta(z|x_i). \tag{2.4}$$

Then, it is easy to verify that the estimation given by $v(\theta; \mathcal{S}) v(\theta; \mathcal{S})^T$ is indeed unbiased:

$$\begin{aligned} &\mathbb{E}_{p_\theta(z|\mathcal{S})} \left[v(\theta; \mathcal{S}) v(\theta; \mathcal{S})^T \right] \\ &\quad = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \mathbb{E}_{p_\theta(z|x_i)} \left[v(\theta; x_i) v(\theta; x_i)^T \right] \\ &\qquad + \frac{1}{|\mathcal{S}|} \sum_{\substack{i,j \in \mathcal{S} \\ i \neq j}} \mathbb{E}_{p_\theta(z|x_i)} \left[v(\theta; x_i) \right] \mathbb{E}_{p_\theta(z|x_j)} \left[v(\theta; x_j)^T \right] \\ &\quad = B(\theta; \mathcal{S}), \end{aligned} \tag{2.5}$$

where the second equality follows from

$$\mathbb{E}_{p_\theta(z|x)} \left[\nabla \log p_\theta(z|x) \right] = \int \nabla p_\theta(z|x) dz = \nabla 1 = 0. \tag{2.6}$$

Using the above estimation, our QNG method employs the iteration

$$\theta_{t+1} = \theta_t - \eta_t V_t^{-1} \nabla F(\theta_t; \mathcal{S}_t), \tag{2.7a}$$

$$V_{t+1} = (1 - c_v) V_t + c_v v(\theta_t; \mathcal{S}_t) v(\theta_t; \mathcal{S}_t)^T, \tag{2.7b}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

where the matrix $V_t \in \mathbb{R}^{n\times n}$ can be viewed as an exponential moving average of the stochastic Fisher and $c_v \in (0,1)$ is a hyper-parameter controlling the decay rate. When the step size η_t is small, the matrix V_t serves as an unbiased estimate of the Fisher $B(\theta_t)$, and in this sense, second-order information is incorporated for overcoming the ill-conditioning issue.

2.1. Limited-memory QNG iteration. The update in (2.7) requires us to update and invert a matrix of size $n \times n$ and its cost is prohibitive when n is large. We deal with this issue using the well-known limited-memory technique. The additive update in (2.7b), however, seems to admit no simple way to invert V_t. A more natural choice is to derive an equivalent multiplicative update which facilitates efficient computation of matrix inverse.

The method starts with factorizing the matrix V_t as $V_t = A_t A_t^T$ and substituting it into (2.7b) for both t and $t+1$:

$$A_{t+1}A_{t+1}^T = (1-c_v)A_tA_t^T + c_v v(\theta_t;\mathcal{S}_t)v(\theta_t;\mathcal{S}_t)^T. \tag{2.8}$$

Defining a vector $q_t \in \mathbb{R}^n$ such that

$$v(\theta_t;\mathcal{S}_t) = A_t q_t \tag{2.9}$$

and pulling out A_t on the right-hand side of (2.8), one gets

$$A_{t+1}A_{t+1}^T = A_t\left((1-c_v)I + c_v q_t q_t^T\right)A_t^T. \tag{2.10}$$

With direct calculation of the square root of the term $(1-c_v)I + c_v q_t q_t^T$, one finally obtains a multiplicative update rule

$$A_{t+1} = A_t K_t, \quad \text{where} \quad K_t = \alpha I + \beta_t q_t q_t^T, \tag{2.11}$$

$\alpha = \sqrt{1-c_v}$, and $\beta_t = \frac{1}{\|q_t\|^2}(\sqrt{1-c_v+\|q_t\|^2 c_v} - \sqrt{1-c_v})$. The transformation matrix K_t is rank one and carries the second-order information explored in the iteration t.

As a next step, the recursive multiplicative update (2.11) will be formulated in a limited-memory manner. By expanding (2.11), we can rewrite A_t in a nonrecursive form

$$A_t = A_1 K_1 K_2 \cdots K_{t-1}. \tag{2.12}$$

The transformation matrix K_j, by definition, carries information no later than the iteration j. Having chosen a memory parameter l, we can restrict the use of past information to the last l matrices $\{K_j\}_{j=t-l}^{t-1}$, since the earlier ones $\{K_j\}_{j=1}^{t-l-1}$, as well as A_1, are likely to contribute little to the descent step at the current iteration t. To implement this idea, we introduce a new matrix $\tilde{A}_t \in \mathbb{R}^{n\times n}$ to approximate A_t by truncating the matrices earlier than the iteration $t-l$:

$$\tilde{A}_t = K_{t-l}K_{t-l+1}\cdots K_{t-1}. \tag{2.13}$$

By doing this, we maintain a limited memory of recent captured second-order information, thereby making it possible to reduce the algorithm complexity. Unlike in L-BFGS or its stochastic counterparts, the initial matrix A_1 is fixed to I. We use no initialization tricks for $\tilde{A}_t$ since they seem to complicate the algorithm while yielding no significant improvement in performance.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

With the matrix $\tilde{A}_t$, we define a limited-memory version of the QNG iteration as

$$\theta_{t+1} = \theta_t - \eta_t \tilde{A}_t^{-T} \tilde{A}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t), \tag{2.14a}$$

$$q_t = \tilde{A}_t^{-1} v(\theta_t; \mathcal{S}_t). \tag{2.14b}$$

Equation (2.14a) implements the limited-memory version of (2.7a) by replacing V_t with $\tilde{A}_t \tilde{A}_t^T$. Explicitly maintaining the Fisher becomes unnecessary; instead, the vector q_t for reconstructing $\tilde{A}_t$ should be calculated and stored.

For a more precise description, first pick a stochastic minibatch $\mathcal{S}_t$ and obtain the gradient estimate $\nabla F(\theta_t; \mathcal{S}_t)$ and the rank-one Fisher estimate $v(\theta_t; \mathcal{S}_t)$. Compute then a temporary variable $u = \tilde{A}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t)$ and perform the descent step $\theta_{t+1} = \theta_t - \eta_t \tilde{A}_t^{-T} u$. Calculate $q_t = \tilde{A}_t^{-1} v(\theta_t; \mathcal{S}_t)$ as the final output and this will be involved in reconstructing the Fisher in the next l iterations.

The limited-memory QNG iteration (2.14) employs three matrix-vector multiplications of two types:

$$\tilde{A}_t^{-1} u = K_{t-1}^{-1} K_{t-2}^{-1} \cdots K_{t-l}^{-1} u, \tag{2.15a}$$

$$\tilde{A}_t^{-T} u = K_{t-l}^{-1} K_{t-l+1}^{-1} \cdots K_{t-1}^{-1} u. \tag{2.15b}$$

The matrix K_j^{-1} can be cheaply obtained using the Sherman–Morrison formula

$$K_j^{-1} = \frac{1}{\sqrt{1-c_v}} I - \left(\frac{1}{\sqrt{1-c_v}} - \frac{1}{\sqrt{1-c_v + c_v \|q_j\|^2}} \right) \frac{q_j q_j^T}{\|q_j\|^2}. \tag{2.16}$$

Therefore, both (2.15a) and (2.15b) can be performed using at most l pairs of dot product and vector addition; their only difference is the sequence of multiplying the matrices $\{K_j\}_{j=t-l}^{t-1}$. We describe in Algorithm 2.1 a unified procedure for computing the multiplication of form $u = K_{i_1}^{-1} K_{i_2}^{-1} \cdots K_{i_\tau}^{-1} v$ in a one-loop recursion. We can obtain $\tilde{A}_t^{-1} u$ by calling this procedure with an ordered set of vectors $\{q_{t-1}, q_{t-2}, \ldots, q_{t-\tau}\}$ while obtaining $\tilde{A}_t^{-T} u$ with $\{q_{t-\tau}, q_{t-\tau+1}, \ldots, q_{t-1}\}$, where $\tau = \min\{l, t\}$. The norm $\|q_t\|$ should be stored together with q_t to save computation time, as the former will be used multiple times in subsequent iterations. Note that we extract the coefficient $\frac{1}{\sqrt{1-c_v}}$ from K_j^{-1} in (2.16) and multiply it back after the recursion (see the difference between (2.16) and line 3 in Algorithm 2.2); this will save $\tau - 1$ vector-scalar multiplications.

Algorithm 2.1 *OneLoopRecursion* for computing $u = K_{i_1}^{-1} K_{i_2}^{-1} \cdots K_{i_\tau}^{-1} v$

Require: Vector v, ordered set of vectors $\{q_{i_1}, q_{i_2}, \ldots, q_{i_\tau}\}$

1: $u = v$
2: **for** $j = i_\tau, i_{\tau-1}, \ldots, i_1$ **do**
3: $\quad u = u - \frac{1}{\|q_j\|^2} \left(1 - \sqrt{\frac{1-c_v}{1-c_v+c_v\|q_j\|^2}} \right) (q_j^T u) q_j$
4: **end for**
5: $u = (1-c_v)^{-\tau/2} u$
6: **return** u

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

2.2. Implementation and computational cost. The pseudocode of the complete method is given in Algorithm 2.2. Here we choose $|\mathcal{S}_t| = b$ for all t. Two core steps in each iteration are the computation of QNG in lines 5–11 and the computation of vector q_t in lines 12–16. The QNG is obtained by first computing $\tilde{A}_t^{-1}\nabla F(\theta_t;\mathcal{S}_t)$ in line 8 and then $\tilde{A}_t^{-T}u$ in line 9, where u is the result of the first multiplication. Both multiplications are performed with vectors $\{q_j\}_{j=t-l}^{t-1}$ in the one-loop recursion. In the case $1 < t < l$ we define $\tau = t-1$ and proceed to use only τ available vectors $\{q_j\}_{j=t-\tau}^{t-1}$. The multiplications can be simply skipped when $t = 1$ given $\tilde{A}_1 = I$. After the descent step, the vector q_t is calculated using $\tilde{A}_t^{-1}v(\theta_t;\mathcal{S}_t)$ and maintained together with its norm. Precisely, it is initialized to $v(\theta_t;\mathcal{S}_t)$ and, if $t > 1$, multiplied by $\tilde{A}_t^{-1}$ which is reconstructed using no more than l previously available vectors $\{q_j\}_{j=t-\tau}^{t-1}$.

Algorithm 2.2 QNG method

Require: Initial solution θ_1, step size sequence $\{\eta_t\}_{t=1}^{\infty}$, minibatch size b, memory parameter l, decay rate $c_v = 1-(s_f)^{\frac{1}{l}}$ where s_f is user-defined

1: **for** $t = 1, 2, \cdots$ **do**
2: Choose a subset $\mathcal{S}_t \subset \{1, 2, \ldots, N\}$ of size b
3: Calculate the stochastic gradient $\nabla F(\theta_t;\mathcal{S}_t)$ as defined in (2.1)
4: Calculate the vector $v(\theta_t;\mathcal{S}_t)$ as defined in (2.4)
5: $u = \nabla F(\theta_t;\mathcal{S}_t)$
6: $\tau = \min(t-1, l)$
7: **if** $t > 1$ **then**
8: $u = OneLoopRecursion(u, \{q_{t-1}, q_{t-2}, \ldots, q_{t-\tau}\})$
9: $u = OneLoopRecursion(u, \{q_{t-\tau}, q_{t-\tau+1}, \ldots, q_{t-1}\})$
10: **end if**
11: $\theta_{t+1} = \theta_t - \eta_t u$
12: **if** $t > 1$ **then**
13: $q_t = OneLoopRecursion(v(\theta_t;\mathcal{S}_t), \{q_{t-1}, q_{t-2}, \ldots, q_{t-\tau}\})$
14: **else**
15: $q_t = v(\theta_t;\mathcal{S}_t)$
16: **end if**
17: Store q_t and $\|q_t\|$
18: **if** $t > l$ **then**
19: Discard q_{t-l} and $\|q_{t-l}\|$
20: **end if**
21: **end for**

A parameter that deserves special treatment is the decay rate c_v. It controls the weight of recently captured second-order information in the matrix $\tilde{A}_t$ and can significantly influence the algorithm performance. However, tuning this parameter can be difficult, as its optimal value seems to be highly correlated to the memory parameter l. To address this, we propose to use a heuristic strategy to transform it to a new one which is much easier to tune. This strategy is based on the partition of the approximate Fisher $\tilde{A}_t\tilde{A}_t^T$ as

$$\tilde{A}_t\tilde{A}_t^T = (1-c_v)^l I + U, \tag{2.17}$$

where $U \in \mathbb{R}^{n\times n}$ is some matrix dependent on the vectors $\{q_j\}_{j=t-l}^{t-1}$. The matrix U is positive semidefinite and has probabilistically upper bounded eigenvalues (see

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Lemma 3.1). Therefore, the QNG direction can be regarded as a standard SGD direction, corrected by the information encoded in the matrix U. The term $(1-c_v)^l$ should be considered as a whole, because it acts as a scaling factor applied to the SGD direction. Hence, we define a scaling parameter s_f as

$$s_f = (1-c_v)^l \tag{2.18}$$

and use it as the input parameter instead of c_v. With a specific s_f, the QNG method explores the SGD direction with a fixed amount of computational effort; changing the memory parameter l only influences the correction imposed by the second-order information. We find this strategy makes the parameter tuning much easier.

Here, we discuss the computational cost of the QNG method. For comparison, we also consider the oLBFGS method [33], a well-known second-order method for large-scale stochastic optimization. First, the QNG method maintains l vectors $\{q_j\}_{j=t-l}^{t-1}$ in memory, so its space complexity is $O(ln)$. On problems with a very large decision space, the memory consumption could be substantial, and this is indeed a general limitation of most second-order methods. The memory usage of QNG is smaller than that of oLBFGS, as the latter requires storing $2l$ vectors of size n. The reason is in oLBFGS the curvature information is explored from the pairs of variable variation and gradient variation and this doubles the memory usage.

The QNG method requires approximately $6ln$ operations[2] in executing the three multiplications; for oLBFGS, the two-loop recursion requires only $4ln$ operations [27]. Hence, the QNG method requires about 50% more computing time in the descent and update steps than oLBFGS. Nevertheless, the extra computations can be compensated by less overhead in capturing the second-order information. To see this, let us consider a classical binary classification task, namely, the logistic regression problem. This problem uses a negative log-likelihood loss with the probabilistic density function given by

$$p_\theta(z|x) = \sigma(z\theta^T x) = \begin{cases} \sigma(\theta^T x) & \text{if } z = 1, \\ 1-\sigma(\theta^T x) & \text{if } z = -1, \end{cases} \tag{2.19}$$

where σ denotes the sigmoid function

$$\sigma(a) = \frac{1}{1+\exp(-a)}. \tag{2.20}$$

Evaluating $p_\theta(z_i|x_i)$ for each sample ξ_i requires approximately n operations and thus the cost of obtaining the stochastic gradient is approximately $2bn$. The oLBFGS method further doubles the gradient evaluations as it has to compute the gradient variation for capturing the curvature information. Thus, it takes approximately $4bn + 4ln$ operations per iteration.

Unlike oLBFGS, the QNG iteration only uses one stochastic gradient but requires one additional vector $v(\theta_t; \mathcal{S}_t)$. It may appear, based on the definition $v(\theta_t; \mathcal{S}_t) = (1/\sqrt{|\mathcal{S}_t|}) \sum_{i\in\mathcal{S}_t} \nabla \log p_\theta(\hat{z}_i|x_i)$ with $\hat{z}_i \sim p_\theta(z|x_i)$, that evaluating $v(\theta_t; \mathcal{S}_t)$ is as expensive as evaluating the stochastic gradient. But this is not the case because evaluating $p_\theta(\hat{z}_i|x_i)$ can resort to evaluating the sigmoid function $\sigma(\theta^T x_i)$ which has already been done when evaluating the stochastic gradient. In other words, the QNG iteration avoids the most expensive part in evaluating the vector $v(\theta_t; \mathcal{S}_t)$, which takes

[2] We count a multiplication and an addition as an operation.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

approximately bn operations. Therefore, the total cost of each QNG iteration is approximately $3bn + 6ln$. This leads to a notable difference in number of required operations of

$$3bn + 6ln \text{ for QNG} \quad \text{versus} \quad 4bn + 4ln \text{ for oLBFGS}. \tag{2.21}$$

In common settings for limited-memory methods, the memory parameter l is usually smaller than 20 and the additional cost of QNG to execute the three-loop procedure is negligible. The minibatch size b, contrarily, can be much larger to facilitate parallelization or noise reduction; in this setting, the QNG iteration is more efficient in exploring the second-order information. That is, when using a relatively large minibatch size, the QNG method can be competitive to or even better than the oLBFGS method in terms of the computing time. For example, with the setting $b = 200$ and $l = 10$, each iteration of QNG reduces about $1/4$ computing time needed by oLBFGS. Similar results regarding the relative cost of the QNG and oLBFGS iterations can be obtained on more complicated problems, such as the multilayer perception training tasks; see Appendix D for details.

3. Convergence. In this section, we analyze the convergence properties of the proposed QNG method. The first goal here is to show that the new method globally converges to a critical point without the assumption on convexity, which is often considered as an advantage of natural gradient methods over other second-order methods. The convergence rate on strong convex problems, under some standard assumptions, is also investigated.

To facilitate the subsequent analyses, we denote the Fisher approximation by $\tilde{V}_t$, i.e.,

$$\tilde{V}_t = \tilde{A}_t \tilde{A}_t^T = K_{t-\tau} \cdots K_{t-1} K_{t-1} \cdots K_{t-\tau}, \quad \text{where } \tau = \min\{t-1, l\}. \tag{3.1}$$

We first show that this matrix has eigenvalues bounded above and away from zero.

LEMMA 3.1. *Let $\tau = \min\{t-1, l\}$. The Fisher approximation $\tilde{V}_t$ satisfies*

$$\alpha^{2l} I \preceq \tilde{V}_t \preceq \prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\| \alpha^{-l} \sqrt{c_v}\right)^2 I \quad \textit{for} \quad t = 1, 2, \ldots, \tag{3.2}$$

where $\alpha = \sqrt{1 - c_v}$ is defined in (2.11).

Proof. See Appendix A. ∎

Lemma 3.1 does not require any assumptions. This means the Fisher approximations are positive definite by design and thus the QNG method is able to handle nonconvexity. On the other hand, the upper bound of the approximate Fisher depends on the vectors $\{v(\theta_t; \mathcal{S}_t)\}$ and one has to restrict these vectors to make sure that the QNG provides a sufficient descent.

We give now some assumptions that will be used in proving global convergence. Here we use $\mathbb{E}_t$ to denote the expectation taken with respect to all randomness at the tth iteration.

Assumption 1. The objective function F is continuously differentiable and is lower bounded by a scalar F_{min}. The gradient ∇F is Lipschitz continuous with constant $L > 0$; namely, for any $a, b \in \mathbb{R}^n$, $\|\nabla F(a) - \nabla F(b)\| \leq L\|a - b\|$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Assumption 2. There exists a constant M_g such that, for all $t \geq 1$,

$$\mathbb{E}_t[\|\nabla F(\theta_t; \mathcal{S}_t)\|^2] \leq M_g^2. \tag{3.3}$$

Assumption 3. The step size sequence is selected as nonsummable but square summable, i.e.,

$$\sum_{t=1}^{\infty} \eta_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty. \tag{3.4}$$

Assumption 4. There exists a constant M_v such that, for all $t \geq 1$,

$$\mathbb{E}_t[\|v(\theta_t; \mathcal{S}_t)\|^2] \leq M_v^2. \tag{3.5}$$

Assumptions 1, 2, and 3 are customary in the analysis of stochastic optimization methods. Both the boundedness of gradient norms and the use of diminishing step sizes are intended to eliminate the adverse effect caused by the random variation. Assumption 4 is specific to QNG and we use it to upper bound the approximate Fisher; without it, the QNG iteration will still produce a descent direction in expectation, but may not be guaranteed to approach the optimum. We show this in the following lemma.

LEMMA 3.2. *Under Assumptions* 1 *and* 2*, the iteration of QNG satisfies the following inequality for all* $t \geq 1$*:*

$$\begin{aligned} &\mathbb{E}_t[F(\theta_{t+1})] - F(\theta_t) \\ &\quad \leq -\eta_t \frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2} + \frac{L}{2}\eta_t^2 \alpha^{-4l} M_g^2. \end{aligned} \tag{3.6}$$

Proof. See Appendix B. ∎

In Lemma 3.2, the bound of the expected per-iteration progress depends on the random vectors $\{v(\theta_t; \mathcal{S}_t)\}$ and therefore we can only get a probabilistically sufficient descent condition. This is the major difference from other second-order methods based on Hessian approximations. The benefit is that it alleviates the need for the positive definiteness of the captured second-order information, allowing QNG to handle nonconvexity without additional mechanisms such as damping or regularization. Using Lemma 3.2, it is sufficient to prove the global convergence property of QNG in probability.

THEOREM 3.3. *Suppose that Assumptions* 1–4 *hold for* $\{\theta_t\}$ *generated by QNG. Then*

$$\lim_{t \to \infty} \|\nabla F(\theta_t)\|^2 = 0 \quad \textit{with probability } 1. \tag{3.7}$$

Proof. We first prove a weaker result,

$$\liminf_{t \to \infty} \|\nabla F(\theta_t)\|^2 = 0 \quad \text{with probability 1.} \tag{3.8}$$

Let $\mathbb{E}[\cdot]$ denote the total expectation taken over all previous randomness, i.e.,

$$\mathbb{E}[F(\theta_t)] = \mathbb{E}_1 \mathbb{E}_2 \ldots \mathbb{E}_{t-1}[F(\theta_t)]. \tag{3.9}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Taking the expectation $\mathbb{E}$ at both sides of (3.6) and summing for $t = 1, 2, \ldots, T$ give

$$\begin{aligned} F_{min} - F(\theta_1) &\leq \mathbb{E}[F(\theta_{T+1})] - F(\theta_1) \\ &\leq -\sum_{t=1}^{T} \eta_t \mathbb{E}\left[\frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau}\left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2}\right] + \frac{L}{2}\alpha^{-4l}M_g^2\sum_{t=1}^{T}\eta_t^2. \end{aligned} \tag{3.10}$$

Let $T \to \infty$ and recall that $\sum_{t=1}^{\infty}\eta_t^2 < \infty$; we have

$$\sum_{t=1}^{\infty} \eta_t \mathbb{E}\left[\frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau}\left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2}\right] < \infty. \tag{3.11}$$

On the other hand, by Assumption 4 and Markov's inequality, we can bound the denominator in (3.11) as

$$\begin{aligned} &\mathbb{P}\left[\prod_{j=1}^{\tau}\left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 \geq Y_t\right] \\ &\leq \max_{1\leq j\leq\tau} \mathbb{P}\left[\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v} \geq Y_t^{1/2\tau}\right] \\ &\leq Y_t^{-1/2\tau} \max_{1\leq j\leq\tau} \mathbb{E}\left[\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right] \\ &\leq Y_t^{-1/2\tau}\left(\alpha + M_v\alpha^{-l}\sqrt{c_v}\right), \end{aligned} \tag{3.12}$$

where $Y_t > 0$ is fixed when a specific t is given. This bound cannot be used directly as it depends on t. To address this, we can choose, for any $\epsilon_1 > 0$, some constant W satisfying

$$W > \max\left\{1, \left(\frac{\epsilon_1}{\alpha + M_v\alpha^{-l}\sqrt{c_v}}\right)^{2l}\right\}. \tag{3.13}$$

Then, using (3.12) and (3.13) and the fact $\tau = \min\{t-1, l\}$, we have

$$\begin{aligned} &\mathbb{P}\left[\prod_{j=1}^{\tau}\left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 \geq W\right] \\ &\leq \mathbb{P}\left[\prod_{j=1}^{\tau}\left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 \geq \left(\frac{\epsilon_1}{\alpha + M_v\alpha^{-l}\sqrt{c_v}}\right)^{2\tau}\right] \\ &\leq \frac{1}{\epsilon_1}, \end{aligned} \tag{3.14}$$

which is a bound (in probability) that is independent of t. We can now proceed to bound the norm of the gradient as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

$$
\begin{aligned}
&\limsup_{t\to\infty} \mathbb{P}\left[\|\nabla F(\theta_t)\|^2 \geq \epsilon\right] \\
&\quad \leq \limsup_{t\to\infty} \mathbb{P}\left[\|\nabla F(\theta_t)\|^2 \geq \epsilon, \prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 \leq W\right] \\
&\qquad + \limsup_{t\to\infty} \mathbb{P}\left[\|\nabla F(\theta_t)\|^2 \geq \epsilon, \prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 > W\right] \\
&\quad \leq \limsup_{t\to\infty} \mathbb{P}\left[\frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2} \geq \frac{\epsilon}{W}\right] \\
&\qquad + \limsup_{t\to\infty} \mathbb{P}\left[\prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2 > W\right] \\
&\quad \leq \frac{1}{\epsilon_1},
\end{aligned}
\tag{3.15}
$$

where in the second inequality we have used the bounds in (3.11) and (3.14) and the condition $\sum_{t=1}^{\infty} \eta_t = \infty$. It follows that the claim (3.8) is true.

The desired result (3.7) can be derived from (3.8) in a method similar to the analysis of Wang et al. [37]. Assume now (3.7) does not hold. According to (3.8), there exist two sequences $\{u_i\}$ and $\{v_i\}$ satisfying $u_i < v_i$ such that, for a given $\epsilon_2 > 0$,

$$
\|\nabla F(\theta_{u_i})\| \geq 2\epsilon_2, \|\nabla F(\theta_{v_i})\| < \epsilon_2, \|\nabla F(\theta_t)\| \geq \epsilon_2, \text{ for } t = u_i + 1, \ldots, v_i - 1. \tag{3.16}
$$

This implies $u_i > v_{i-1}$ and hence the sets $\{t\}_{t=u_i}^{v_i-1}$ are disjoint. Then, from (3.11), we obtain

$$
\begin{aligned}
\infty &> \sum_{i=1}^{\infty} \sum_{t=u_i}^{v_i-1} \eta_t \mathbb{E}\left[\frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2}\right] \\
&> \epsilon_2^2 \sum_{i=1}^{\infty} \sum_{t=u_i}^{v_i-1} \eta_t \frac{1}{\mathbb{E}\left[\prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2\right]}.
\end{aligned}
\tag{3.17}
$$

It follows, using (3.14), that $\infty > \epsilon_2^2 \sum_{i=1}^{\infty} \sum_{t=u_i}^{v_i-1} \eta_t$ with probability 1 and therefore

$$
\lim_{i\to\infty} \sum_{t=u_i}^{v_i-1} \eta_t = 0 \quad \text{with probability 1.} \tag{3.18}
$$

Consider then the expected norm of descent steps. Using Lemma 3.1, Assumption 2, and Jensen's inequality yields

$$
\mathbb{E}\|\theta_{t+1} - \theta_t\| \leq \sqrt{\mathbb{E}\|\theta_{t+1} - \theta_t\|^2} \leq \eta_t \alpha^{-2l} M_g, \tag{3.19}
$$

which, together with (3.18) and the convexity of $\|\cdot\|$, implies

$$
\mathbb{E}\|\theta_{v_i} - \theta_{u_i}\| \leq \sum_{t=u_i}^{v_i-1} \mathbb{E}\|\theta_{t+1} - \theta_t\| \leq \alpha^{-2l} M_g \sum_{t=u_i}^{v_i-1} \eta_t. \tag{3.20}
$$

We finally conclude from (3.18) and (3.20) that

$$
\lim_{i\to\infty} \|\theta_{v_i} - \theta_{u_i}\| = 0 \quad \text{with probability 1,} \tag{3.21}
$$

a contradiction to (3.16). Hence, the claim in (3.7) is valid. ∎

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

We complement the above global convergence property with a characterization of the expected convergence rate that we detail below.

THEOREM 3.4. *Consider the QNG method as defined in Algorithm* 2.2 *and suppose that Assumptions* 1–4 *hold. Further assume*

- *the objective function F is strongly convex with constant $\mu > 0$;*
- *the norms of the vectors $\{v(\theta_t; \mathcal{S}_t)\}$ are deterministically bounded by $M_v > 0$, i.e., $\|v(\theta_t; \mathcal{S}_t)\| \leq M_v$ for all $t \geq 1$;*
- *the step size is chosen as $\eta_t = \frac{\eta_1 T_1}{T_1 + t - 1}$ with parameter $T_1 > 0$ where*

$$\eta_1 T_1 > \frac{C_1}{2\mu} \quad \text{and} \quad C_1 = \max\left\{1, \left(\alpha + M_v \alpha^{-l}\sqrt{c_v}\right)^{2l}\right\}.$$

Then the expected difference between the objective value $F(\theta_t)$ and the optimal objective F_* satisfies

$$\mathbb{E}[F(\theta_t)] - F_* \leq \frac{T_1}{T_1 + t - 1} \max\left\{\frac{\frac{L}{2}\eta_1^2 T_1 \alpha^{-4l} M_g^2}{\eta_1 T_1 \frac{2\mu}{C_1} - 1}, F(\theta_1) - F_*\right\}, \tag{3.22}$$

where $\mathbb{E}$ denotes the total expectation defined in (3.9).

Proof. See Appendix C. ∎

Remarks. The $O(1/t)$ convergence rate established in Theorem 3.4 is typical of stochastic optimization methods. Although this is no better than SGD, the QNG method can often enjoy the improvements in convergence time, as illustrated in the numerical experiments in section 4. This sublinear rate in theory, compared to deterministic methods which often exhibit a linear convergence rate, is due to the existence of noise. Notice that in QNG there exist two kinds of noises, one in the stochastic gradient estimates $\nabla F(\theta; \mathcal{S})$ and the other one in the vectors $v(\theta; \mathcal{S})$ for approximating the Fisher. While both can influence the algorithm performance, these noises have different roles in determining the convergence rate. To see this, consider the expected per-iteration progress of the objective values given in Lemma 3.2. The vectors $v(\theta; \mathcal{S})$ appear as a scaling factor of $\|\nabla F(\theta)\|$. The bound of the noise in stochastic gradients, M_g, appears in the term that is independent of $\|\nabla F(\theta)\|$ and this means such noise may inhibit the convergence even when θ is near the optimum. If there were no noise in the gradients (e.g., when $M_g \propto \|\nabla F(\theta)\|$), then one could easily obtain linear convergence with a fixed step size; in this case, the noise of the vectors $v(\theta; \mathcal{S})$ does not influence the convergence rate up to a factor. On the contrary, even when we could eliminate the noise in the Fisher approximation, it is still impossible to obtain a convergence rate faster than $O(1/t)$ if the gradient computation is noisy [1]. These observations imply that the QNG method can be implemented with the stochastic variance reduction techniques [13] to achieve a better convergence rate and we leave it as a future study.

4. Experimental results.

To verify the theoretical results, we evaluate the performance of QNG for solving a number of classification problems arising in machine learning.

4.1. Experimental setup.

We first consider two convex problems, logistic regression and softmax regression, and two nonconvex problems, nonlinear least squares classification and nonconvex support vector machine. All these problems have objective functions taking the form of (1.1) and (1.2). In particular, we consider using the ℓ_2 regularization

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$$R(\theta) = \frac{\lambda}{2}\|\theta\|^2, \tag{4.1}$$

where the parameter λ is set to 10^{-6} throughout the experiment. The loss functions of these problems are defined as follows:

• *Logistic regression*:

$$\ell(h(\theta, x), z) = -\log \sigma(z\theta^T x), \quad z \in \{-1, 1\}, \tag{4.2}$$

where $\sigma(\cdot)$ is the sigmoid function given in (2.20).

• *Softmax regression*:

$$\ell(h(\theta, x), z) = -\sum_{k=1}^{c} \mathbb{I}\{z = k\} \log \frac{\exp(\theta_k^T x)}{\sum_{j=1}^{c} \exp(\theta_j^T x)}, \quad z \in \{1, \ldots, k\}, \tag{4.3}$$

where c is the number of classes.

• *Nonlinear least squares classification*:

$$\ell(h(\theta, x), z) = \left(\tanh(\theta^T x) - z\right)^2, \quad z \in \{-1, 1\}, \tag{4.4}$$

where $\tanh(\cdot)$ is defined as

$$\tanh(a) = \frac{1 - \exp(-2a)}{1 + \exp(-2a)}. \tag{4.5}$$

• *Nonconvex support vector machine*:

$$\ell(h(\theta, x), z) = 1 - \tanh(z\theta^T x), \quad z \in \{-1, 1\}. \tag{4.6}$$

Both logistic regression and softmax regression adopt the negative log-likelihood loss. The nonlinear least squares classification problem, which implements a single-layer perception model [29], can also be viewed as adopting a Gaussian likelihood loss. The proposed QNG method is applicable on these problems because they all lead to a standard maximum likelihood learning task defined on a certain statistical manifold.

The nonconvex support vector machine was originally introduced in [23]. This problem does not model a probability distribution over the outputs and there seems to be no simple way to convert it into a standard density estimation task. Nevertheless, the loss function qualifies the margin of a simple classifier $\mathrm{sgn}(\theta^T x)$ which is a surrogate of error probability [7] and therefore does have implicit statistical meanings. To see this, from the fact $1 - \tanh(z\theta^T x) \propto 1 - \sigma(2z\theta^T x)$ and $\log(\sum_{i=1}^{N} \sigma(2z_i\theta^T x_i)) \geq \sum_{i=1}^{N} \log \sigma(2z_i\theta^T x_i)$, we can estimate a probability density $p_\theta(z|x) = \sigma(2z\theta^T x)$ which upper bounds the objective function near its optimum. This motivates using QNG to solve the nonconvex support vector machine in the manifold admitted by the probability model $p_\theta(z|x) = \sigma(2z\theta^T x)$. The goal of choosing this problem is to show the effectiveness of QNG even in the heuristic setting.

In total, eight datasets are used in the numerical study.[3] Their statistics are briefly summarized in Table 1. The rcv1, news20, real-sim, and gisette datasets are binary and used for logistic regression, nonlinear least squares classification, and nonconvex support vector machine. The other four contain multiclass outputs and are used for softmax regression. We have scaled all samples in each dataset to ensure that they have an average ℓ_2 norm 1. In all these datasets, 70% of the data are used for training while the remaining 30% are used for testing.

[3] All the datasets can be downloaded at www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

TABLE 1
Statistics of the used datasets.

Dataset	n	N	# classes
rcv1	47236	677399	2
news20	1355191	19996	2
real-sim	20958	72309	2
gisette	5000	6000	2
covtype	54	581012	7
mnist	784	60000	10
SensIT	100	78823	3
protein	367	17766	3

We compare the performance of QNG with that of SGD and oLBFGS. For nonlinear least squares classification and the nonconvex support vector machine, the oLBFGS method is equipped with explicit damping techniques to tackle nonconvexity [37, 27]. For all these algorithms, the minibatch size is set to $b = \lfloor \sqrt{N} \rfloor$ and the step size sequence is of the form

$$\eta_t = \frac{\eta_1 T_1}{T_1 + t - 1} \tag{4.7}$$

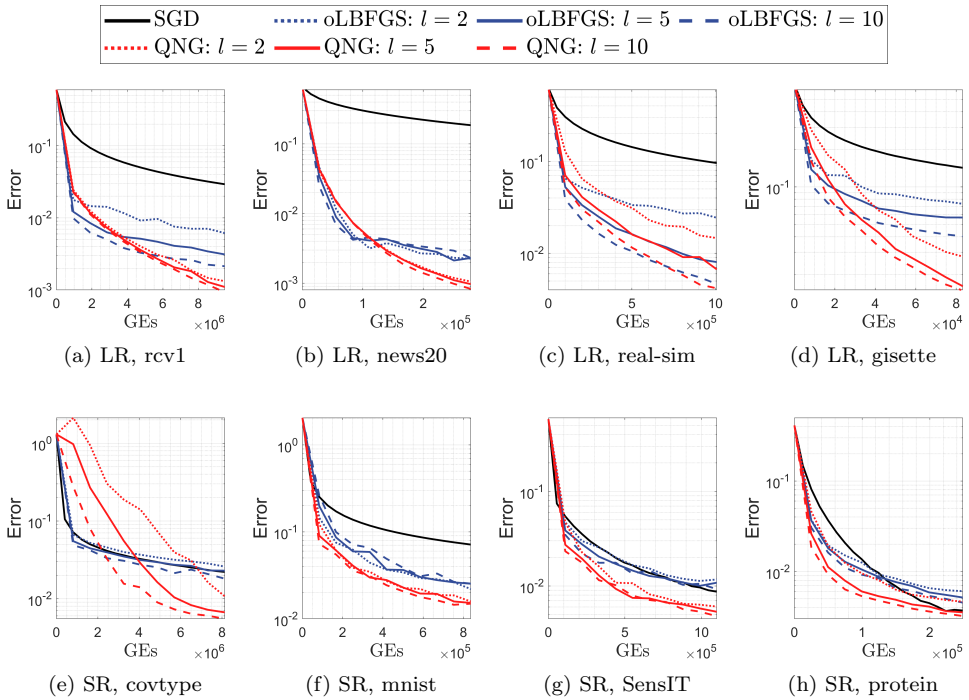
as in Theorem 3.4. A grid search is performed to find the optimal step size parameters η_1 and T_1, as well as learning parameter s_f in QNG, from candidate values $\eta_1 \in \{1, 2, 4\}$, $T_1 \in \{10^1, 10^3, 10^5\}$, and $s_f \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. In the grid search, the memory parameter l in QNG and oLBFGS is fixed to 5, but other values such as 2 and 10 are also considered in the subsequent experiments. All algorithms are initialized at the vector of zeros and run with a budget of $20N$ gradient evaluations. For a fair competition, one evaluation of the vector $v(\theta_t; \mathcal{S}_t)$ in QNG is counted as b stochastic gradient evaluations, considered as expensive as one minibatch gradient. On all test cases, the algorithms are run independently 20 times and the median results are reported.

Apart from comparing the algorithms on individual test instances,[4] we also report their overall performance in a set of test instances using the performance profiles [10]. The profile of an algorithm is the curve of the fraction of its solved test instances (denoted by $\rho(\tau)$) versus the amount of allocated computational budget (denoted by τ). The computational budget is measured by the ratio of the required running time or gradient evaluations to that required by the best performer. We say an algorithm can solve a test instance if its obtained objective function value f' satisfies $f_0 - f' > 0.05(f_0 - f_*)$, where f_0 is the initial objective function value and f_* is the best value obtained among all algorithms in all independent runs. An algorithm with high values of $\rho(\tau)$ or one that is located at the top right of the figure is preferable. The profile is plotted to show the training and generalization performance individually, where the objective functions are respectively the training loss and the test loss.

4.2. Numerical results on convex problems. Figure 1 shows the convergence curves in the training on the two convex problems. In most cases, the two second-order methods, QNG and oLBFGS, converge much faster than the first-order method, SGD, demonstrating the power of utilizing second-order information. QNG and oLBFGS behave similarly in general; this is because both logistic regression and

[4] A test instance refers to a certain problem run on a certain dataset.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) LR, rcv1 (b) LR, news20 (c) LR, real-sim (d) LR, gisette

(e) SR, covtype (f) SR, mnist (g) SR, SensIT (h) SR, protein

FIG. 1. *Comparison of QNG with SGD and oLBFGS on logistic regression (LR) and softmax regression (SR). Plots display the training error over the number of gradient evaluations (GEs). Both QNG and oLBFGS are tested with the memory parameter l varying in* $\{2, 5, 10\}$.

softmax regression belong to the canonical generalized linear models and therefore the Hessian and Fisher coincide [16, Chapter 10.8]. The difference in the behaviors of QNG and oLBFGS is due to the different ways in exploring the second-order information. QNG is slightly slower than oLBFGS in the initial phase but tends to produce lower objective function values after a few epochs.

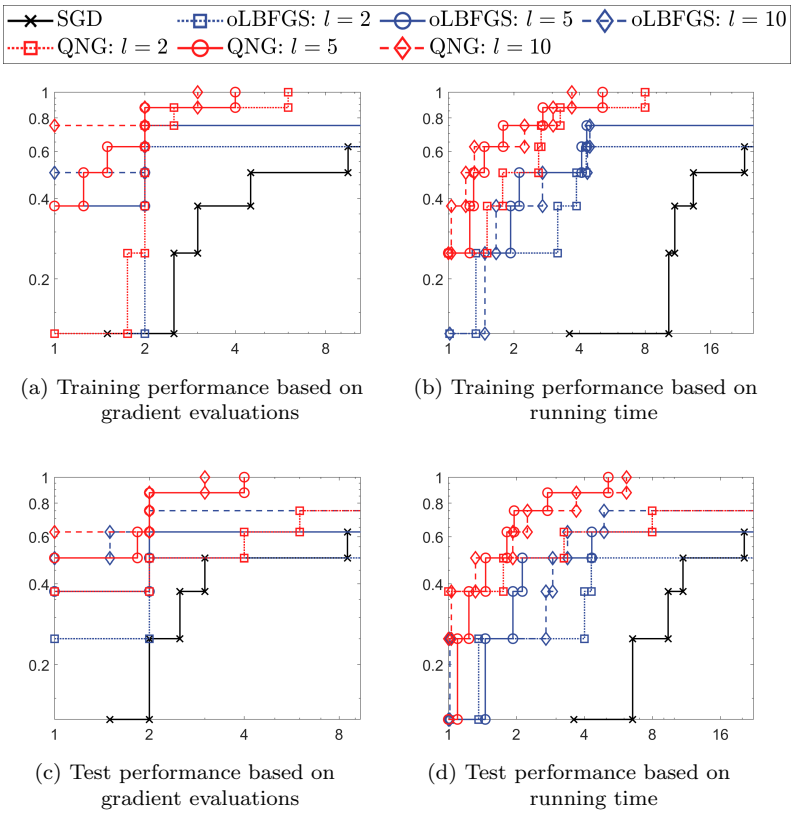
Similar to that for oLBFGS, the memory parameter l has a significant effect on the performance of QNG. Increasing l from 2 to 5 and further to 10 consistently improves QNG, but the benefit is not obvious in every case. It suggests that, just like in the limited-memory approximation to the Hessian, the optimal memory size in the limited-memory approximation to the Fisher is problem-dependent.

Figure 2 provides the performance profiles obtained on the convex problems. For both gradient evaluations and running time and for both training and test, the curves of QNG mostly lie to the left of others, indicating its superior performance. The profiles of QNG and oLBFGS overlap for small τ when measured in terms of the gradient evaluations. However, the overlaps become unapparent when the profiles are measured with running time. This implies in current settings QNG has less per-iteration computing time than oLBFGS. On the other hand, setting l to 2 is sufficient to solve all problems for training but fails on more than 20% of problems for testing. We may thus conclude that the memory size has a considerable impact on the generalization performance of QNG.

4.3. Numerical results on nonconvex problems. To ensure that oLBFGS can handle nonconvex problems, we consider using two damping techniques in updating its Hessian approximation: (1) the classical damping technique proposed by Powell

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) Training performance based on gradient evaluations

(b) Training performance based on running time

(c) Test performance based on gradient evaluations

(d) Test performance based on running time

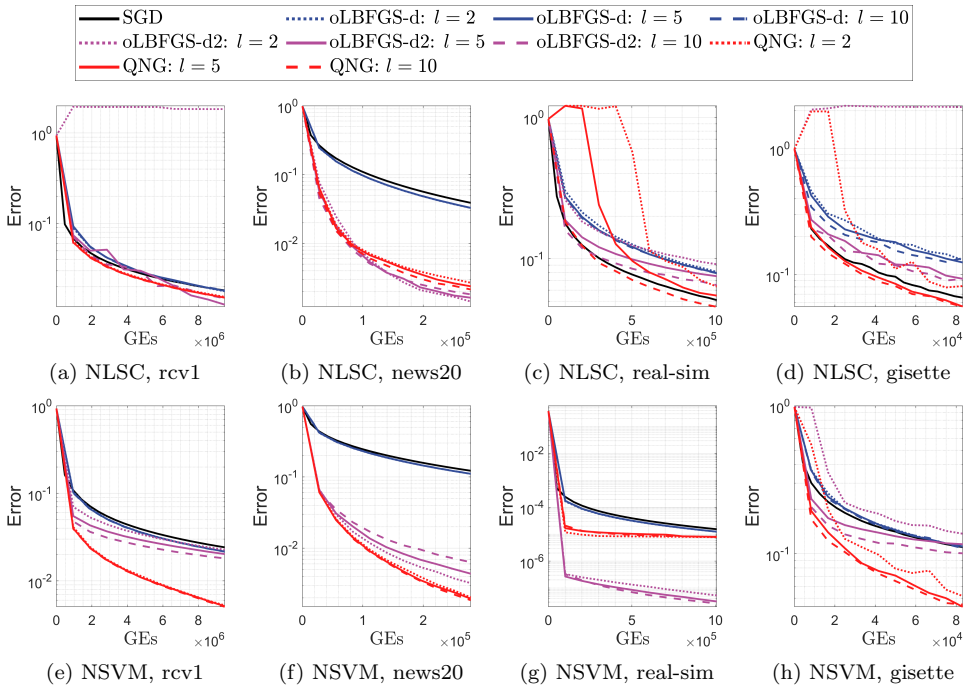
FIG. 2. *Performance profiles of SGD, oLBFGS, and QNG on convex problems. Numerical results are obtained on logistic regression over rcv1/news20/real-sim/gisette and softmax regression over covtype/mnist/SensIT/protein. In* (a) *and* (b) *the performance is measured using training loss and in* (c) *and* (d) *using test loss.*

(see Chapter 11 in [27]) and (2) a very recent one proposed by Wang et al. [37]. The corresponding algorithms are denoted by oLBFGS-d and oLBFGS-d2, respectively. Powell's method uses an additional two-loop recursion to correct negative curvatures and thus doubles the computing cost in every iteration. Wang et al.'s method avoids this issue by preregularizing the gradient variation but introduces a new regularization parameter. In this experiment, the regularization parameter of oLBFGS-d2 is chosen from $\{1, 0.1, 0.01\}$ using a grid search procedure.

Figure 3 shows the convergence behaviors of SGD, oLBFGS-d, oLBFGS-d2, and QNG on nonconvex problems. Unlike in the convex cases, oLBFGS does not show clear superiority over SGD when equipped with the Powell's damping technique. Wang et al.'s method improves Powell's in most cases, but the corresponding algorithm, oLBFGS-d2, does not show obvious advantage over SGD. QNG exhibits consistent performance on both convex and nonconvex problems; in most cases, it converges faster than SGD, oLBFGS-d, and oLBFGS-d2. Again, we observe that the effect of the memory size in QNG is problem-dependent: in a few cases the setting $l = 2$ causes divergence in the initial phase. This is possibly because that the step size is tuned with $l = 5$; a more careful parameter tuning procedure may alleviate this problem. In many other cases the performance is insensitive to this memory parameter and setting l to 5 seems to be sufficient.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) NLSC, rcv1 (b) NLSC, news20 (c) NLSC, real-sim (d) NLSC, gisette

(e) NSVM, rcv1 (f) NSVM, news20 (g) NSVM, real-sim (h) NSVM, gisette

FIG. 3. *Comparison of QNG with SGD, oLBFGS-d, and oLBFGS-d2 on nonlinear least squares classification (NLSC) and nonconvex support vector machine (NSVM). Plots display the training error over the number of gradient evaluations (GEs). Both QNG and oLBFGS-d are tested with the memory parameter l varying in $\{2, 5, 10\}$.*

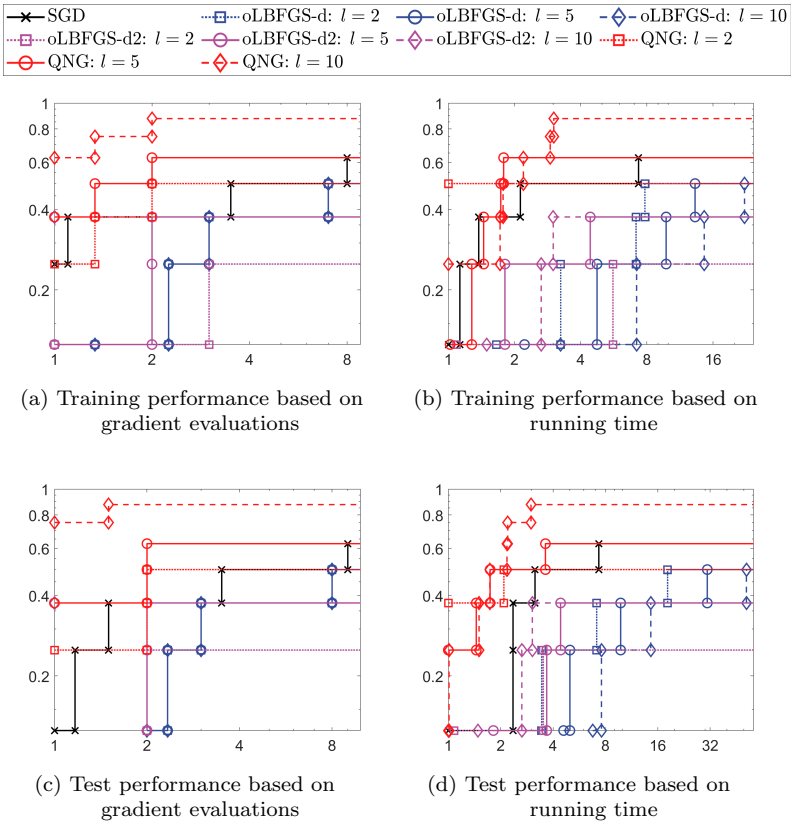
Figure 4 further gives a visual comparison of the four algorithms using profiles. QNG with $l = 10$ is superior to all the other algorithms in both training and testing. QNG with $l = 2$ or 5 is competitive with oLBFGS with $l = 10$ when the computational budget is limited (say, $\tau \leq 2$), but requires many fewer computations in achieving $\rho(\tau) \geq 60\%$. In general, QNG is very efficient in terms of both gradient evaluations and running time.

4.4. Robustness to small minibatches. In the experiments presented in subsections 4.2 and 4.3, we used a relatively large minibatch size, $b = \lfloor \sqrt{n} \rfloor$. The intention in choosing this setting is threefold: (1) to accelerate the process through parallelization such as auto-vectorization and auto-multithreading, (2) to offset the additional costs of computing the matrix-vector products, and (3) to reduce the additional noise introduced in the rank-one approximation of the Fisher. Now we test QNG with a much smaller minibatch size and investigate whether its performance is robust in this setting.

Figure 5 reports the convergence curves of SGD, oLBFGS, and QNG on logistic regression with $b = 20$. In this setting, the overhead of performing the quasi-natural gradient descent in QNG is not negligible. Nevertheless, the numeric results suggest that this overhead seems to be well compensated by its fast convergence performance. QNG is superior to or competitive with SGD in all cases. It, for instance, achieves better results within shorter running time on news20, real-sim, and gisette. On the other hand, when comparing these plots with Figure 1, one immediately finds that reducing the minibatch size degrades the performance of oLBFGS and makes it even

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) Training performance based on gradient evaluations

(b) Training performance based on running time

(c) Test performance based on gradient evaluations

(d) Test performance based on running time

FIG. 4. *Performance profiles of SGD, oLBFGS-d, oLBFGS-d2, and QNG on nonconvex problems. Numerical results are obtained on nonlinear least squares and the nonconvex support vector machine, both over the rcv1, news20, real-sim, and gisette datasets. In* (a) *and* (b) *the performance is measured using training loss and in* (c) *and* (d) *using test loss.*

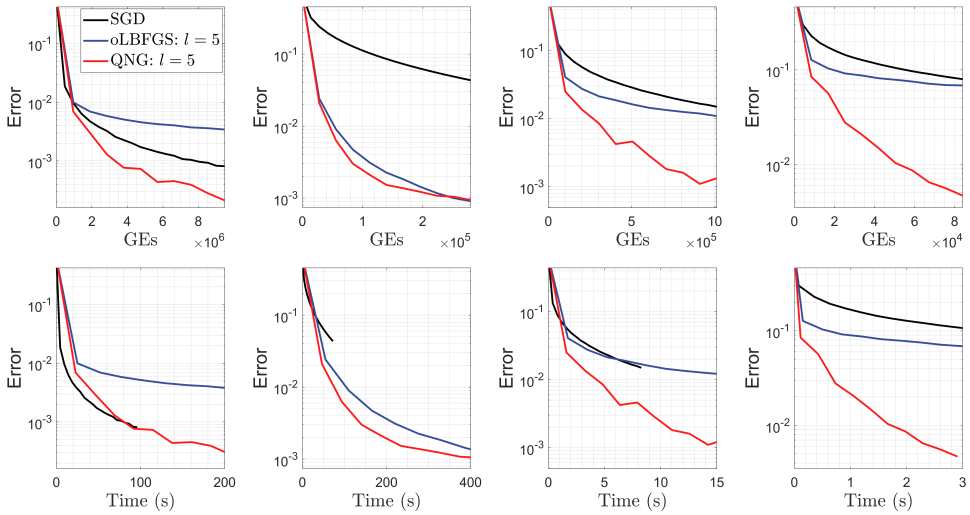
worse than SGD in certain cases. The performance degradation of oLBFGS with a small minibatch has been observed in [33] and the authors empirically verified that to overcome this one has to use a large memory parameter l which further increases the per-iteration cost. QNG does not suffer from this issue, exhibiting a better robustness than oLBFGS for small minibatch sizes. Note, however, that the advantage of QNG over SGD becomes insignificant on the largest dataset rcv1, which implies that using such a small minibatch may not be sufficient to produce informative Fisher approximations. This experiment suggests that QNG has a great robust against small minibatches but a relative large minibatch size is in general preferable.

4.5. More experiments on multilayer perceptron training. We further consider a more challenging problem, the training of multilayer perceptron (MLP) classifiers. The MLP model used in this experiment has a fully connected hidden layer with 100 neurons, where the tanh function (4.5) is used as the activation function. The last layer of the model is a softmax function defined in (4.3). The experiments are performed on the four multiclass datasets introduced in subsection 4.1.

We choose SGD and oLBFGS used in the above experiments for comparison. We additionally include two state-of-the-art stochastic optimization methods, namely ADAM [30] and SignSGD [6]. ADAM is a well-known algorithm for training neural networks which uses a diagonal Fisher approximation as a preconditioner. SignSGD

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) rcv1 (b) news20 (c) real-sim (d) gisette

FIG. 5. *Performance of SGD, oLBFGS, and QNG on logistic regression in the setting of* $b = 20$. *Plots display the training error over the number of gradient evaluations (GEs) and the running time (Time).*

takes the sign of gradient as its descent direction, with the aim of improving the robustness against noise. Both ADAM and SignSGD exploit landscape curvature information to accelerate the search, and in this sense, they are analogous to QNG and oLBFGS. The difference is that, in ADAM and SignSGD, the descent directions are adjusted coordinatewise; therefore, their effectiveness largely depends on the problem characteristics (e.g., sparsity of the gradient and eigenspectrum of the Hessian).

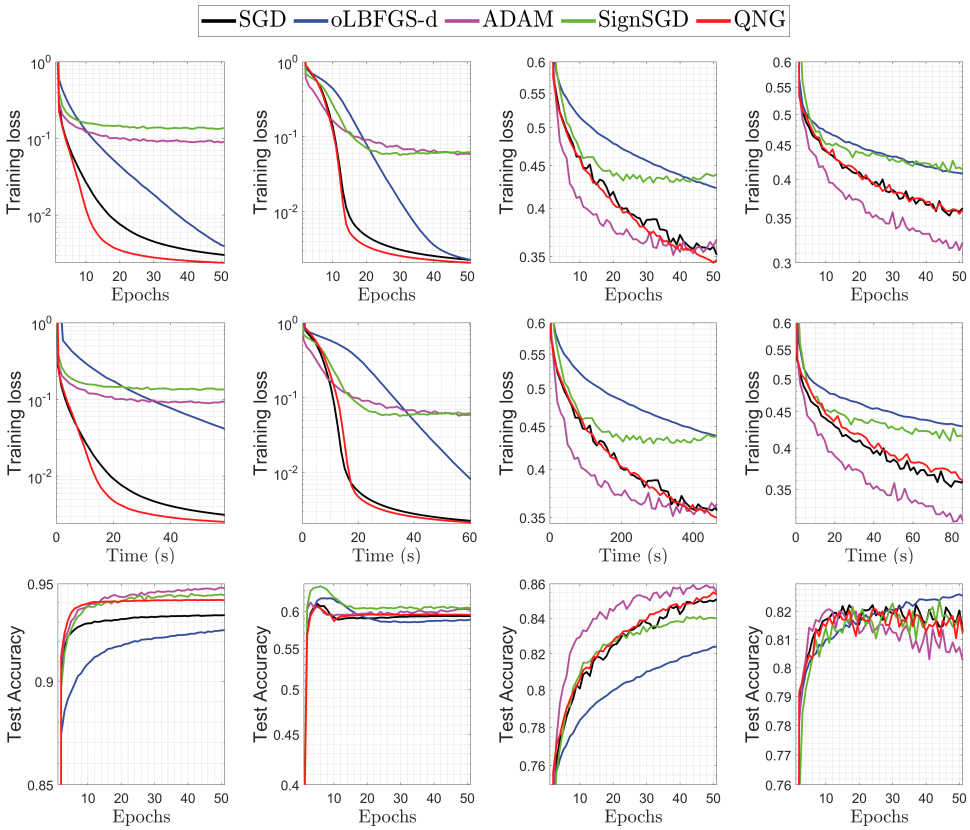
All considered algorithms use a constant step size tuned from candidate values $\{2^{-6}, \ldots, 2^{-1}\}$ using a grid search. For ADAM, the learning rate of the momentum and the learning rate for adapting the variance are set respectively to 0.9 and 0.99. The parameter s_f in QNG is fixed to 0.1. To handle nonconvexity, we implement oLBFGS with the Powell's damping technique [27, Chapter 11] and the method is denoted by oLBFGS-d. The minibatch size is set to $b = \lfloor \sqrt{N} \rfloor$. To improve the generalization performance of the MLP model, we adopt the ℓ_2 regularization defined in (4.1) and set the parameter λ to 10^{-6}.

In QNG, after evaluating the gradient $\nabla F(\theta_t; \mathcal{S}_t)$, the vector $v(\theta_t; \mathcal{S}_t)$ used for approximating the Fisher can be computed through an additional backward pass. Thus, each iteration of QNG takes one forward pass and two backward passes, so it is computationally cheaper than oLBFGS, which requires two forward passes and two backward passes. The implementation details as well as the per-iteration complexity analysis are given in Appendix D.

Figure 6 shows the performance comparison between QNG and the other four methods. Here we use two different methods for measuring the per-iteration computational cost: the number of epochs (in the first row) and the exact CPU computing time (in the second row). From the convergence curves of training loss versus epochs, it is observed that QNG performs clearly better than SignSGD and oLBFGS-d on all four datasets. QNG is also competitive with SGD on SensIT while reaching better final results on all three other datasets. The relative advantage of QNG can also be

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) mnist (b) protein (c) covtype (d) SensIT

FIG. 6. *Training MLP classifiers on the* (a) *mnist,* (b) *protein,* (c) *covtype, and* (d) *SensIT datasets. The first row shows the training loss versus the number of epochs; the second row shows the training loss versus the computing time; the last row shows the prediction accuracy on test data versus the number of epochs.*

observed when the computational cost is measured using the CPU computing time; it implies its additional cost in approximating the Fisher can be partially compensated by the faster convergence. ADAM seems to reach different solutions on mnist, protein, and SensIt, as its performance in training is different from that in testing (see the last row). For example, it achieves the best training performance on SensIt but exhibits the worst prediction accuracy in testing. In general, due to the diagonalized Fisher modeling method, the effectiveness of ADAM may vary according to the problem characteristics, which has also been observed in existing studies [5]. QNG, on the contrary, achieves consistent performance on all four datasets.

5. Related work. Optimization methods inspired by natural gradients have attracted much attention recently in the machine learning community. Except in a few simple cases, natural gradients do not have an easy-to-compute expression and this evokes extensive research interest in dealing with the computational issue. Most existing methods adopt an adaptive scheme for approximating the Fisher which improves the efficiency and robustness when N is large [4, 28, 32, 34, 14, 18, 38, 36]. They typically maintain a metric for the underlying statistical manifold and use stochastic Fisher estimates to update it, and in this way, efficiently estimating the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Fisher becomes the key design step. Early work in [4, 28] focused on incremental online learning, resembling the QNG method in the special case of $b = 1$. Stochastic Fisher in this setting is rank one and this avoids the computational issue but cannot benefit from parallelization or using minibatches. Later works such as [32, 34, 39] explore the low-rank structure of the Fisher by using Gauss–Newton approximation or incremental principal component analysis. They have not eliminated the need for matrix factorization or inversion and the time complexity may scale quadratically or even cubically with the underlying dimension. Our QNG method differs from those methods in that it employs the limited-memory rank-one update rule to approximate the inverse Fisher.

An alternative for simplifying the calculation is to extract the sparse or singular pattern of the Fisher [22, 11, 3, 12, 17]. Very recently, there was a jump in interest in using this idea for deep learning, probably due to the sparse and singular structures abounding in neural networks. For example, the Kronecker-factored approximate curvature (K-FAC) family of deep networks [22, 11] utilizes blockwise approximations and achieves faster convergence than well-tuned SGD in practice. A common feature of these methods is that their major ingredients (e.g., damping, initialization, and approximation) are specific to neural networks and hence they are not general-propose solvers.

In [20] the author proposed a limited-memory method for modeling covariance matrices in the context of high-dimensional black-box optimization, which appears to be similar to our QNG method in technical details. However, the implementation in this paper is much more simple as we only approximate the matrix factor for the Fisher, whereas in [20] both the matrix factor and its inverse are required to be reconstructed.

6. Concluding remarks. In this paper we present a quasi-natural gradient method, QNG, for solving stochastic optimization problems where the search space admits a statistical manifold. QNG resembles classical quasi-Newton methods in approximating the Fisher information matrix but employs a novel limited-memory implementation that may be more efficient in the large minibatch setting. Coupled with this limited-memory technique, a rank-one unbiased method is proposed to estimate the stochastic Fisher, which avoids complicated matrix operations and makes the time complexity scale linearly with the memory length. We establish global convergence of the algorithm on smooth but probably nonconvex problems and obtain a sublinear convergence rate on strongly convex problems. We carry out numerical simulations on several machine learning tasks and show that QNG is superior to or at least competitive with SGD and oLBFGS on convex problems while exhibiting better robustness on nonconvex problems. Although the presented implementation only achieves a sublinear convergence rate, the theoretical analysis suggests that the Fisher approximation can be uniformly bounded in probability. Therefore, our algorithm is compatible with state-of-the-art variance reduction techniques to achieve a faster convergence.

Appendix A. Proof of Lemma 3.1. First, regarding the definition of $\tilde{V}_t$ in (3.1), it will be convenient to define some intermediate results as

$$\begin{cases} \tilde{V}_t^{[0]} = I, \\ \tilde{V}_t^{[j]} = K_{t-j} \tilde{V}_t^{[j-1]} K_{t-j}, \ j = 1, \dots, \tau. \end{cases} \tag{A.1}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Then, $\tilde{V}_t = \tilde{V}_t^{[\tau]}$. By the definition of $\{K_j\}$ in (2.11), we have, for all $t \geq 1$ and $j \in \{1, \ldots, \tau\}$, that $\tilde{V}_t^{[j]}$ is positive definite. This allows us to proceed with the inverse matrix $(\tilde{V}_t^{[j]})^{-1}$.

Let $\lambda_1[\cdot]$ denote the maximum eigenvalue of a specific matrix. It follows from (A.1) that

$$\lambda_1[(\tilde{V}_t^{[j]})^{-1}] = \lambda_1[K_{t-j}^{-1}(\tilde{V}_t^{[j-1]})^{-1}K_{t-j}^{-1}] \leq \lambda_1[(\tilde{V}_t^{[j-1]})^{-1}](\lambda_1[K_{t-j}^{-1}])^2. \tag{A.2}$$

According to the definition of $\{K_j\}$ in (2.11),

$$\lambda_1[K_j^{-1}] = \lambda_1[(\alpha I + \beta_j q_j q_j)^{-1}] = \alpha^{-1}.$$

Substituting this into (A.2) and recalling $\alpha < 1$ we can conclude that

$$\lambda_1[(\tilde{V}_t^{[j]})^{-1}] \leq \lambda_1[(\tilde{V}_t^{[j-1]})^{-1}]\alpha^{-2}$$

and thus

$$\lambda_1[(\tilde{V}_t^{[\tau]})^{-1}] \leq \alpha^{-2\tau} \leq \alpha^{-2l}. \tag{A.3}$$

It gives the lower bound $\tilde{V}_t \succeq \alpha^{2l} I$.

Similarly, by (2.11) and (A.1), we can derive an upper bound for $\tilde{V}_t^{[j]}$ as

$$\begin{aligned}
\lambda_1[\tilde{V}_t^{[j]}] &\leq \lambda_1[\tilde{V}_t^{[j-1]}](\lambda_1[K_{t-j}])^2 \\
&= \lambda_1[\tilde{V}_t^{[j-1]}]\left(\alpha + \beta_{t-j}\|q_{t-j}\|^2\right)^2 \\
&= \lambda_1[\tilde{V}_t^{[j-1]}]\left(\alpha + \sqrt{1 - c_v + \|q_{t-j}\|^2 c_v} - \sqrt{1 - c_v}\right)^2 \\
&\leq \lambda_1[\tilde{V}_t^{[j-1]}]\left(\alpha + \|q_{t-j}\|\sqrt{c_v}\right)^2.
\end{aligned} \tag{A.4}$$

The second inequality uses $\sqrt{a+b} - \sqrt{a} \leq \sqrt{b}$.

By the definition of q_j given in (2.14b), we have

$$\|q_j\|^2 = \|\tilde{A}_j^{-1} v(\theta_j; \mathcal{S}_j)\|^2 = v(\theta_j; \mathcal{S}_j)^T \tilde{V}_j^{-1} v(\theta_j; \mathcal{S}_j) \leq \|v(\theta_j; \mathcal{S}_j)\|^2 \lambda_1[\tilde{V}_j^{-1}]. \tag{A.5}$$

Substituting (A.5) into (A.4) and then applying the bound provided in (A.3) we obtain

$$\lambda_1[\tilde{V}_t] = \lambda_1[\tilde{V}_t^{[\tau]}] \leq \prod_{j=1}^{\tau} \left(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v}\right)^2. \tag{A.6}$$

This completes the proof.

Appendix B. Proof of Lemma 3.2. The proof is standard in stochastic optimization and is given below for completeness.

The Lipschitz continuity of the gradient in Assumption 1 implies that

$$\begin{aligned}
F(\theta_{t+1}) - F(\theta_t) &\leq \nabla F(\theta_t)^T(\theta_{t+1} - \theta_t) + \frac{L}{2}\|\theta_{t+1} - \theta_t\|^2 \\
&= -\eta_t \nabla F(\theta_t)^T \tilde{V}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t) + \frac{L}{2}\eta_t^2 \|\tilde{V}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t)\|^2 \\
&\leq -\eta_t \nabla F(\theta_t)^T \tilde{V}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t) + \frac{L}{2}\eta_t^2 \lambda_1[\tilde{V}_t^{-1}]^2 \|\nabla F(\theta_t; \mathcal{S}_t)\|^2 \\
&\leq -\eta_t \nabla F(\theta_t)^T \tilde{V}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t) + \frac{L}{2}\eta_t^2 \alpha^{-4l} \|\nabla F(\theta_t; \mathcal{S}_t)\|^2,
\end{aligned} \tag{B.1}$$

where in the last inequality we use the result from Lemma 3.1.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Taking expectation with respect to $\mathcal{S}_t$, we have

$$\begin{aligned}
&\mathbb{E}_t[F(\theta_{t+1})] - F(\theta_t) \\
&\quad \le -\eta_t \mathbb{E}_t[\nabla F(\theta_t)^T \tilde{V}_t^{-1} \nabla F(\theta_t; \mathcal{S}_t)] + \frac{L}{2}\eta_t^2 \alpha^{-4l} \mathbb{E}_t[\|\nabla F(\theta_t; \mathcal{S}_t)\|^2] \\
&\quad \le -\eta_t \nabla F(\theta_t)^T \tilde{V}_t^{-1} \nabla F(\theta_t) + \frac{L}{2}\eta_t^2 \alpha^{-4l} M_g^2 \\
&\quad \le -\eta_t \frac{\|\nabla F(\theta_t)\|^2}{\prod_{j=1}^{\tau}(\alpha + \|v(\theta_{t-j}; \mathcal{S}_{t-j})\|\alpha^{-l}\sqrt{c_v})^2} + \frac{L}{2}\eta_t^2 \alpha^{-4l} M_g^2.
\end{aligned} \tag{B.2}$$

The second inequality uses $\mathbb{E}_t[\nabla F(\theta_t; \mathcal{S}_t)] = \nabla F(\theta_t)$ and Assumption 2. The third inequality follows directly from Lemma 3.1. This gives the desired bound.

Appendix C. Proof of Theorem 3.4.

According to Lemma 3.1, the constant C_1 upper bounds the maximal eigenvalue of the approximate Fisher $\tilde{V}_t$. We can rewrite Lemma 3.2, using the μ-strong convexity, as

$$\mathbb{E}_t[F(\theta_{t+1})] - F(\theta_t) \le -\eta_t \frac{2\mu}{C_1}(F(\theta_t) - F_*) + \frac{L}{2}\eta_t^2 \alpha^{-4l} M_g^2. \tag{C.1}$$

Substituting $\eta_t = \frac{\eta_1 T_1}{T_1 + t - 1}$ and rearranging (C.1) yields

$$\begin{aligned}
\mathbb{E}_t[F(\theta_{t+1})] - F_* \le &\left(1 - \frac{\eta_1 T_1}{T_1 + t - 1}\frac{2\mu}{C_1}\right)(F(\theta_t) - F_*) \\
&+ \frac{L}{2}\left(\frac{\eta_1 T_1}{T_1 + t - 1}\right)^2 \alpha^{-4l} M_g^2.
\end{aligned} \tag{C.2}$$

The desired conclusion (3.22) holds trivially when $t = 1$. Now we prove the case $t > 1$ by induction. Assume (3.22) holds for some $t \ge 1$ and define

$$\rho = T_1 \max\left\{\frac{\frac{L}{2}\eta_1^2 T_1 \alpha^{-4l} M_g^2}{\eta_1 T_1 \frac{2\mu}{C_1} - 1}, F(\theta_1) - F_*\right\};$$

then from (C.2) we have

$$\begin{aligned}
\mathbb{E}[F(\theta_{t+1}) - F_*] &\le \left(1 - \frac{\eta_1 T_1}{T_1 + t - 1}\frac{2\mu}{C_1}\right)\frac{1}{T_1 + t - 1}\rho \\
&\quad + \frac{L}{2}\left(\frac{\eta_1 T_1}{T_1 + t - 1}\right)^2 \alpha^{-4l} M_g^2 \\
&= \frac{(T_1 + t - 1) - 1}{(T_1 + t - 1)^2}\rho + \frac{1 - \eta_1 T_1 \frac{2\mu}{C_1}}{(T_1 + t - 1)^2}\rho + \frac{\frac{L}{2}\eta_1^2 T_1^2 \alpha^{-4l} M_g^2}{(T_1 + t - 1)^2} \\
&\le \frac{1}{T_1 + t}\rho + \frac{(1 - \eta_1 T_1 \frac{2\mu}{C_1})\rho + \frac{L}{2}\eta_1^2 T_1^2 \alpha^{-4l} M_g^2}{(T_1 + t - 1)^2}.
\end{aligned} \tag{C.3}$$

Using the assumption $\eta_1 T_1 > \frac{C_1}{2\mu}$ and the definition of ρ yields (3.22).

Appendix D. Implementation details on multilayer perceptron classifiers. In this section, we discuss how to implement QNG on a more complicated

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

problem, the training of MLP models. Specifically, we consider a J-layer MLP classifier formulated as

$$\begin{cases} a^{[1]} = x \in \mathbb{R}^{n_1}, \\ a^{[j]} = \sigma_h\left(W^{[j]}a^{[j-1]} + b^{[j]}\right) \in \mathbb{R}^{n_j}, j = 2, \ldots, J-1, \\ a^{[J]} = \sigma_o\left(W^{[J]}a^{[J-1]} + b^{[J]}\right) \in \mathbb{R}^{n_J}, \end{cases} \tag{D.1}$$

where $W^{[j]} \in \mathbb{R}^{n_{j-1} \times n_j}$ is the weight matrix, $b^{[j]} \in \mathbb{R}^{n_j}$ is the bias vector, σ_h is the elementwise activation function in the hidden layers, and σ_o is the activation function in the output layer. At the jth layer, n_j is the number of neurons, and $a^{[j]}$ is the layer's output formulated as an n_j-dimensional vector. The above MLP model can be parameterized by

$$\theta = \left(W^{[2]}, b^{[2]}, \ldots, W^{[J]}, b^{[J]}\right)$$

and its dimension is on the order of $O(\sum_{j=2}^{J} n_{j-1}n_j)$. We assume that the last layer is a softmax classifier satisfying

$$a^{[J]} \propto \exp\left(W^{[J]}a^{[J-1]} + b^{[J]}\right) \text{ and } \sum_{k=1}^{n_J} a_k^{[J]} = 1,$$

where $a_k^{[J]}$ denotes its kth output and the function exp is performed elementwise. The vector $a^{[J]}$ is thus the prediction of the MLP model, i.e., $h(\theta, x) = a^{[J]}$, and the corresponding loss function is

$$\ell\left(h(\theta, x), z\right) = -\sum_{k=1}^{n_J} \mathbb{I}\left\{z = k\right\} \log a_k^{[J]},$$

where $z \in \{1, \ldots, n_J\}$ is the index of target label. For simplicity, denote $\mathcal{I}(z)$ as the n^J-dimensional indicator vector corresponding to z, i.e., the kth element of $\mathcal{I}(z)$ is 1 if and only if $z = k$. Simple calculations reveal that the gradient of the loss depends on the prediction error and can be written as

$$\nabla\ell\left(h(\theta, x), z\right) = g(\mathcal{I}(z) - a^{[J]})$$

for some function g that can be evaluated using the chain rule. Hence, when given a minibatch $\mathcal{S}$ of training samples, the gradient can be approximated by

$$\nabla F(\theta; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla\ell\left(h(\theta, x_i), z_i\right) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} g\left(\mathcal{I}(z_i) - a^{[J]}\right), \tag{D.2}$$

where z_i is the target output associated with the input x_i. Meanwhile, as the above loss function has already been in the negative log-likelihood form, for approximating the Fisher, we can compute the vector $v(\theta; \mathcal{S})$ as

$$v(\theta; \mathcal{S}) = \frac{1}{\sqrt{|\mathcal{S}|}} \sum_{i \in \mathcal{S}} \nabla\ell\left(h(\theta, x_i), \hat{z}_i\right) = \frac{1}{\sqrt{|\mathcal{S}|}} \sum_{i \in \mathcal{S}} g\left(\mathcal{I}(\hat{z}_i) - a^{[J]}\right), \tag{D.3}$$

where $\hat{z}_i$ is randomly drawn from the prediction distribution

$$\mathbb{P}\left[\hat{z}_i = k\right] = a_k^{[J]}. \tag{D.4}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Comparing (D.2) and (D.3), we know both $\nabla F(\theta;\mathcal{S})$ and $v(\theta;\mathcal{S})$ require evaluating the prediction model $h(\theta, x_i)$ for all $i \in \mathcal{S}$. In the context of MLP training, it implies that, when computing $\nabla F(\theta;\mathcal{S})$ and $v(\theta;\mathcal{S})$ simultaneously, the forward pass for each training sample can be performed only once. The only difference in computing $\nabla F(\theta;\mathcal{S})$ and $v(\theta;\mathcal{S})$ is that the target output in the former is from the training set while in the latter it is drawn from the prediction model. This difference will only influence the evaluation of the prediction error, which is the input of the backward pass. That is, we can reuse the back-propagation procedure to compute $v(\theta;\mathcal{S})$ without writing additional code. Detailed steps are given below:

1. Forward pass: input x_i and compute $a^{[1]}, \ldots, a^{[J]}$ in order using (D.1).
2. Error evaluation-I: input z_i and compute $e_i = \mathcal{I}(z_i) - a^{[J]}$.
3. Backward pass-I: input e_i and use back-propagation to compute $g(e_i)$.
4. Error evaluation-II: draw $\hat{z}_i$ from the distribution in (D.4) and compute $\hat{e}_i = \mathcal{I}(\hat{z}_i) - a^{[J]}$.
5. Backward pass-II: input $\hat{e}_i$ and use back-propagation to compute $g(\hat{e}_i)$.

From (D.3), it is clear that $g(\hat{e}_i)\, g(\hat{e}_i)^T$ is an unbiased estimator of the Fisher (up to a constant scalar). Repeating these steps over the whole minibatch $\mathcal{S}$ gives both $\nabla F(\theta;\mathcal{S})$ and $v(\theta;\mathcal{S})$.

Now we can proceed to analyze the time cost of QNG. It is found that in each iteration QNG requires one forward pass and two backward passes for each training sample. Denote the time complexity of forward and backward passes by $\mathcal{T}_f$ and $\mathcal{T}_b$, respectively. Further denote $\mathcal{T}_l$ as the time complexity of one loop procedure used in the limited-memory scheme. Then, the per-iteration time cost of QNG is about $|\mathcal{S}|\,(\mathcal{T}_f + 2\mathcal{T}_b) + 3\mathcal{T}_l$.

We further consider oLBFGS for comparison. The per-iteration cost of oLBFGS is about $2|\mathcal{S}|\,(\mathcal{T}_f + \mathcal{T}_b) + 2\mathcal{T}_l$, as it involves one fewer loops for computing the descent direction but one more forward pass for each training sample. Usually, we have $\mathcal{T}_f = \Theta(\sum_{j=2}^{J} n_{j-1}n_j), \mathcal{T}_b = \Theta(\sum_{j=2}^{J} n_{j-1}n_j), \mathcal{T}_l = \Theta(l\sum_{j=2}^{J} n_{j-1}n_j)$, and it is reasonable to assume the minibatch size $|\mathcal{S}|$ to be significantly larger than the memory parameter l. Then, we can compare QNG and oLFBGS as

$$\frac{\text{per-iteration cost of QNG}}{\text{per-iteration cost of oLBFGS}} \approx \frac{|\mathcal{S}|\,(\mathcal{T}_f + 2\mathcal{T}_b) + 3\mathcal{T}_l}{2|\mathcal{S}|\,(\mathcal{T}_f + \mathcal{T}_b) + 2\mathcal{T}_l} \approx \frac{\mathcal{T}_f + 2\mathcal{T}_b}{2\mathcal{T}_f + 2\mathcal{T}_b}.$$

The relative cost of QNG and oLBFGS depends on how to implement the MLP models; if the forward and backward passes have the similar computation cost, then QNG can reduce about 1/4 time required by oLBFGS. Also note that, due to the nonconvexity of MLP models, oLBFGS should always be implemented with damping mechanisms; this makes oLBFGS take more computing time.

REFERENCES

[1] A. Agarwal, P. L. Bartlett, P. Ravikumar, and M. J. Wainwright, *Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization*, IEEE Trans. Inform. Theory, 58 (2012), pp. 3235–3249, https://doi.org/10.1109/TIT.2011.2182178.

[2] S.-i. Amari, *Natural gradient works efficiently in learning*, Neural Comput., 10 (1998), pp. 251–276, https://doi.org/10.1162/089976698300017746.

[3] S.-i. Amari, R. Karakida, and M. Oizumi, *Fisher information and natural gradient learning in random deep networks*, in Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 694–702.

[4] S.-i. Amari, H. Park, and K. Fukumizu, *Adaptive method of realizing natural gradient learning for multilayer perceptrons*, Neural Comput., 12 (2000), pp. 1399–1409, https://doi.org/10.1162/089976600300015420.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[5] L. Balles and P. Hennig, *Dissecting Adam: The sign, magnitude and variance of stochastic gradients*, in Proceedings of the International Conference on Machine Learning, 2018, pp. 404–413.
[6] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, *signSGD: Compressed optimisation for non-convex problems*, in Proceedings of the International Conference on Machine Learning, July 2018, pp. 560–569.
[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, in Proceedings of the 5th Annual Workshop on Computational Learning Theory, Pittsburgh, 1992, ACM Press, pp. 144–152, https://doi.org/10.1145/130385.130401.
[8] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Rev., 60 (2018), pp. 223–311, https://doi.org/10.1137/16M1080173.
[9] R. Byrd, S. Hansen, J. Nocedal, and Y. Singer, *A stochastic quasi-newton method for large-scale optimization*, SIAM J. Optim., 26 (2016), pp. 1008–1031, https://doi.org/10.1137/140954362.
[10] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213, https://doi.org/10.1007/s101070100263.
[11] R. Grosse and J. Martens, *A Kronecker-factored approximate fisher matrix for convolution layers*, in Proceedings of the 33rd International Conference on International Conference on Machine Learning, Vol. 48, New York, 2016, pp. 573–582.
[12] W. Guo, Y. Ong, Y. Zhou, J. R. Hervas, A. Song, and H. Wei, *Fisher information matrix of unipolar activation function-based multilayer perceptrons*, IEEE Trans. Cybernet., 49 (2019), pp. 3088–3098, https://doi.org/10.1109/TCYB.2018.2838680.
[13] R. Johnson and T. Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, in Proceedings of the 26th International Conference on Neural Information Processing Systems, Vol. 1, Red Hook, NY, 2013, Curran Associates, pp. 315–323.
[14] M. E. Khan and D. Nielsen, *Fast yet simple natural-gradient descent for variational inference in complex models*, in Proceeding of the International Symposium on Information Theory and Its Applications, 2018, pp. 31–35, https://doi.org/10.23919/ISITA.2018.8664326.
[15] F. Kunstner, P. Hennig, and L. Balles, *Limitations of the empirical Fisher approximation for natural gradient descent*, in Advances in Neural Information Processing Systems 32, Curran Associates, 2019, pp. 4158–4169.
[16] K. Lange, *Optimization*, 2nd ed., Springer Texts Statist. 95, Springer-Verlag, New York, 2013, https://doi.org/10.1007/978-1-4614-5838-8.
[17] Z. Liao, T. Drummond, I. Reid, and G. Carneiro, *Approximate Fisher information matrix to characterize the training of deep neural networks*, IEEE Trans. Pattern Analysis Machine Intelligence, 42 (2020), pp. 15–26, https://doi.org/10.1109/TPAMI.2018.2876413.
[18] W. Lin, M. E. Khan, and M. Schmidt, *Fast and simple natural-gradient variational inference with mixture of exponential-family approximations*, in Proceeding of the International Conference on Machine Learning, 2019, pp. 3992–4002.
[19] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Program., 45 (1989), pp. 503–528, https://doi.org/10.1007/BF01589116.
[20] I. Loshchilov, *LM-CMA: An alternative to L-BFGS for large-scale black box optimization*, Evolutionary Computation, 25 (2017), pp. 143–171, https://doi.org/10.1162/EVCO_a_00168.
[21] J. Martens, *New Insights and Perspectives on the Natural Gradient Method*, arXiv:1412.1193 [cs, stat], 2017, https://arxiv.org/abs/1412.1193.
[22] J. Martens and R. Grosse, *Optimizing neural networks with Kronecker-factored approximate curvature*, in Proceedings of the International Conference on Machine Learning, 2015, pp. 2408–2417.
[23] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, *Boosting algorithms as gradient descent*, in Advances in Neural Information Processing Systems 12, S. A. Solla, T. K. Leen, and K. Müller, eds., MIT Press, Cambridge, MA, 2000, pp. 512–518.
[24] A. Mokhtari, Alej, and R. Ribeiro, *Global convergence of online limited memory BFGS*, J. Mach. Learn. Res., 16 (2015), pp. 3151–3181.
[25] K. P. Murphy, *Machine learning: A probabilistic perspective*, Adapt. Comput. Mach. Learn., MIT Press, Cambridge, MA, 2012.
[26] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comput., 35 (1980), pp. 773–773, https://doi.org/10.1090/S0025-5718-1980-0572855-7.
[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Ser. Oper. Res., Springer, New York, 2nd ed., 2006.
[28] H. Park, S. I. Amari, and K. Fukumizu, *Adaptive natural gradient learning algorithms for various stochastic models*, Neural Networks, 13 (2000), pp. 755–764, https://doi.org/10.1016/S0893-6080(00)00051-4.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[29] Š. Raudys, *Evolution and generalization of a single neurone:* I. *Single-layer perceptron as seven statistical classifiers*, Neural Networks, 11 (1998), pp. 283–296, https://doi.org/10.1016/S0893-6080(97)00135-4.

[30] S. J. Reddi, S. Kale, and S. Kumar, *On the convergence of Adam and beyond*, in Proceeding of the 6th International Conference on Learning Representations, Vancouver, BC, 2018.

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

[32] N. L. Roux, P.-a. Manzagol, and Y. Bengio, *Topmoumoute online natural gradient algorithm*, in Advances in Neural Information Processing Systems 20, Curran Associates, 2008, pp. 849–856.

[33] N. N. Schraudolph, J. Yu, and S. Günter, *A Stochastic Quasi-Newton Method for Online Convex Optimization*, in Proceedings of the 11th International Conference on Artificial Intelligence and Statistics, 2007, pp. 436–443.

[34] K. Sun and F. Nielsen, *Relative Fisher information and natural gradient for learning large modular models*, in Proceedings of the International Conference on Machine Learning, 2017, pp. 3289–3298.

[35] S. Sun, Z. Cao, H. Zhu, and J. Zhao, *A survey of optimization methods from a machine learning perspective*, IEEE Trans. Cybernet., 50 (2020), pp. 3668–3681, https://doi.org/10.1109/TCYB.2019.2950779.

[36] D. Tang and R. Ranganath, *The variational predictive natural gradient*, in Proceedings of the International Conference on Machine Learning, 2019, pp. 6145–6154.

[37] X. Wang, S. Ma, D. Goldfarb, and W. Liu, *Stochastic quasi-Newton methods for nonconvex stochastic optimization*, SIAM J. Optim., 27 (2017), pp. 927–956, https://doi.org/10.1137/15M1053141.

[38] G. Zhang, J. Martens, and R. B. Grosse, *Fast convergence of natural gradient descent for over-parameterized neural networks*, in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, 2019, pp. 8082–8093.

[39] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse, *Noisy natural gradient as variational inference*, in Proceedings of the International Conference on Machine Learning, 2018, pp. 5852–5861.

[40] G. P. Zhang, *Neural networks for classification: A survey*, IEEE Trans. Systems Man Cybernet C, 30 (2000), pp. 451–462, https://doi.org/10.1109/5326.897072.

Downloaded 07/07/22 to 156.146.45.175 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.