# A Stochastic modified limited memory BFGS for training deep neural networks

**Abstract.** In this work we study stochastic quasi-Newton methods for solving the nonlinear and non-convex optimization problems arising in the training of deep neural networks. We consider the limited memory BFGS (L-BFGS) update in the framework of a trust-region approach. We provide an almost comprehensive overview of recent improvements in quasi-Newton based training algorithms, such as accurate selection of the initial Hessian approximation, efficient solution of the trust region sub-problem with a direct method in high accuracy and an overlap sampling strategy to assure stable quasi-Newton updating by computing gradient differences based on this overlap. We provide a comparison of the standard L-BFGS method with a variant of this algorithm based on a modified secant condition which is theoretically shown to provide an increased order of accuracy in the approximation of the curvature of the Hessian. In our experiments, both quasi-Newton updates exhibit comparable performances. We include results showing the performance of the proposed optimizers in the training of two neural networks for image classification of the well known benchmark datasets MNIST and CIFAR10. Our results show that with a fixed computational time budget the proposed quasi-Newton methods provide comparable or better testing accuracy than the state of the art first-order optimizer Adam.

**Keywords:** Quasi-Newton methods, Limited memory BFGS, Trust region, Stochastic optimization, Deep neural networks

## 1 Introduction

Deep learning has become the leading technique for solving large-scale machine learning problems. After a prolonged slow start, the advent of higher computational power and the introduction of GPU computing, have made possible the training of neural networks with a high number of layers that have shown impressive efficacy in image classification tasks, natural language processing and text analytic, speech recognition and reinforcement learning among other fields. Deep Learning problems are often posed as highly nonlinear and often non-convex unconstrained optimization problems. For instance, in image classification using a training dataset $\{(x_i, y_i)\}_{i=1}^N$ in $C$ classes with input $x_i \in \mathbb{R}^n$ and target $y_i \in \mathbb{R}$, a deep neural network training refers to solving an empirical risk minimization (ERM) problem that can be formulated as follows:

$$\min_{w \in \mathbb{R}^n} F(w) := \frac{1}{N} \sum_{i=1}^{N} f_i(w) \tag{1}$$

where $w \in \mathbb{R}^n$ is the vector of trainable parameters, $N$ is the number of observations in the training dataset and $f_i(w) := f(w; x_i, y_i)$ is a loss function quantifying the prediction error for the $i$th observation of the training dataset.

Finding an efficient optimization algorithm for (1) has attracted many researchers and a number of algorithms have been proposed both in the machine learning and optimization literature. Since in large-scale machine learning problems (i.e. large $n$ and $N$) the computation of the loss function $F(w)$ and the gradient $\nabla F(w)$ is expensive and the computation of the true Hessian $\nabla^2 F(w)$ is not practical,

Stochastic first-order methods have been widely used in many DL applications due to their low per-iteration cost, optimal complexity, easy implementation and proven efficiency in practice. The preferred method is the stochastic gradient descent (SGD) method [32,7], and its variance-reduced [12,19,33] and adaptive [13,20] variants. However, these methods due to use of only first-order gradient information come with several issues such as relatively-slow convergence, highly sensitivity to the choice of hyper-parameter (e.g., step-length and batch size). This issue has been addressed in recent works as [20] where the tuning of the hyper-parameters is accomplished automatically. First-order methods can also find some difficulties in escaping saddle points [36], and exhibit limited benefits of parallelism due to their usual implementation with small mini-batches [23].

On the other hand, second order methods can often find good minima in fewer steps due to their use of curvature information. The main second order method incorporating the inverse Hessian matrix is Newton's method [29] that computes the next update step by

$$w_{k+1} = w_k - \eta \nabla^2 F(w_k)^{-1} \nabla F(w_k).$$

However, Newton's method presents serious computational and memory usage challenges involved in the computation of the Hessian. Moreover, using exact Hessians will result in algorithms that produce sequences moving towards saddle points, as Newton's method encourages rapid local convergence towards any stationary point regardless of the curvature [11,22].

Quasi-Newton and Hessian-free methods are two techniques aimed at incorporating second order information whithout computing and storing the true Hessian matrix. Hessian-free methods attempt to find an approximate Newton direction $\nabla^2 F(w_k)^{-1} \nabla F(w_k)$ using conjugate gradient methods [26,4]. Alternatively, quasi-Newton methods and their limited memory variants [29] attempt to combine the speed of Newtons method and the scalability of first-order methods. In fact, they construct Hessian approximations using only gradient information and exhibit superlinear convergence. Quasi-Newton and stochastic quasi-Newton methods to solve large nonconvex optimization problems arising in deep learning have been recently extensively considered [14,3,5,6,2,31,30].

In this work we consider a limited memory variant of BFGS (L-BFGS), one of the most popular quasi-Newton updates in Broyden's class. We consider a stochastic variant obtained by subsampling. We study also a modified

L-BFGS update obtained through a modified secant condition which is theoretically shown to provide an increased order of accuracy in the approximation of the curvature of the Hessian. Both quasi-Newton methods are used in a trust-region framework. We provide an almost comprehensive overview of recent improvements in quasi-Newton based training algorithms, such as accurate selection of the initial Hessian approximation, efficient solution of the trust-region subproblem with a direct method in high accuracy and an overlap sampling strategy to assure stable quasi-Newton updating by computing gradient differences based on this overlap. We examine the behaviour of the studied quasi-Newton methods in the training of deep neural networks in a supervised learning application, image classification, and provide a comparison with a state of the art first-order method such as Adam.

This paper is organized as follows. We provide an overview of the (limited memory) BFGS method in Section 2. In Section 3 we introduce a modified L-BFGS update obtained by imposing a different secant condition. We describe the use of the modified L-BFGS method in a trust-region framework and its stochastic variant in Sections 4 and 5, respectively. Numerical result are reported in Section 6. Finally, some of the conclusions of this study are included in Section 7.

## 2 An overview on the L-BFGS update

### 2.1 The BFGS update

The BFGS update as Hessian approximation have the following general form

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \quad k = 0, 1, \ldots, \tag{2}$$

and satisfies the *standard secant condition*

$$B_{k+1} s_k = y_k, \tag{3}$$

where $s_k = p_k$ and $y_k = \nabla F(w_t) - \nabla F(w_k)$. The vector $p_k$ is the search direction at iteration $k$ and can be obtained in many different ways, for instance, inside a trust-region framework [10] which proposes a trial point

$$w_t = w_k + p_k. \tag{4}$$

The BFGS updates (2) using only gradient information to incorporate curvature information generate symmetric positive definite matrices, i.e. $B_{k+1} \succ 0$, whenever the initial approximation $B_0 = \gamma_k I$ has the same property and the curvature condition $s_k^T y_k > 0$ holds.

### 2.2 The L-BFGS update and its compact form

For large-scale optimization problems, the limited-memory BFGS (denoted by L-BFGS) would be more efficient. In practice, only a limited collection of the

recent pairs, $\{s_j, y_j\}_{j=0}^{r-1}$ where $r \ll n$, is stored in the following low rank (at most $r$) matrices:

$$S_k := \left[s_{k-r} \ \dots \ s_{k-1}\right], \qquad Y_k := \left[y_{k-r} \ \dots \ y_{k-1}\right], \quad k = 1, 2, \dots. \tag{5}$$

Using (5), the L-BFGS matrix $B_k$ (2) can be represented in the following compact form [29]

$$B_k = B_0 + \Psi_k M_k \Psi_k^T, \tag{6}$$

where

$$\Psi_k = \left[B_0 S_k \ Y_k\right], \qquad M_k = \begin{bmatrix} -S_k^T B_0 S_k & -L_k \\ -L_k^T & D_k \end{bmatrix}^{-1}. \tag{7}$$

In (7), matrices $L_k$, $U_k$ and $D_k$ are respectively the strictly lower triangular part, the strictly upper triangular part and the diagonal part of the following matrix splitting

$$S_k^T Y_k = L_k + D_k + U_k. \tag{8}$$

### 2.3 The initialization of the L-BFGS update

The initial matrix $B_0$ is often set to some multiple of the identity matrix. A heuristic and conventional method to choose this multiple is

$$\gamma_k = \frac{y_{k-1}^T y_{k-1}}{y_{k-1}^T s_{k-1}} := \gamma_k^h. \tag{9}$$

The quotient of (9) is an approximation to an eigenvalue of $\nabla^2 F(w_k)$ and appears to be the most successful method, in practice, to generate initial Hessian approximations [29]. However, in a non-convex DL optimization, the choice of $\gamma_k$ should be carefully operated to avoid the introduction of false negative curvature [14,30]. To this end, an extra condition can be imposed on $\gamma_k$ to avoid $p_k^T B_k p_k < 0$ while $p_k^T \nabla^2(w_k) p_k > 0$. The hyper-parameter $\gamma_k$ is selected in $(0, \hat{\lambda})$ where $\hat{\lambda}$ is the smallest eigenvalue of the following generalized eigenvalue problem

$$(L_k + D_k + L_k^T)u = \lambda S_k^T S_k u, \tag{10}$$

with $L_k$ and $D_k$ defined in (8). If $\hat{\lambda} \leq 0$, then $\gamma_k$ can be set to $\gamma_k^h$.

## 3 A modified L-BFGS update

A modified BFGS update, and a consequently modified L-BFGS algorithm, can be proposed by rewriting (3) as a *modified secant condition*

$$B_{k+1} s_k = y_k^*, \tag{11}$$

where $(s_k, y_k^*)$ gives better curvature information than $(s_k, y_k)$ for updating $B_{k+1}$. Therefore, in a similar fashion as described in the previous section, a modified L-BFGS update can be constructed by using $y_k^*$ in place of $y_k$.

Let $\psi_k = (F_k - F_{k+1}) + (g_k + g_{k+1})^T s_k$. In [35], the vector $y_k^*$ was constructed as

$$y_k^* = y_k + \frac{\psi_k}{\|s_k\|^2} s_k. \tag{12}$$

Definition (12) together with (11) provides more accurate curvature information. In fact, it can be proved that

$$\begin{aligned} s_k^T(\nabla^2 F(w_{k+1})s_k - y_k^*) &= \frac{1}{3} s_k^T (T_{k+1}s_k)s_k + O(\|s_k\|^4), \\ s_k^T(\nabla^2 F(w_{k+1})s_k - y_k) &= \frac{1}{2} s_k^T (T_{k+1}s_k)s_k + O(\|s_k\|^4), \end{aligned} \tag{13}$$

where $T_{k+1}$ is the tensor of $F$ at $w_{k+1}$ in the Taylor series as

$$F_k = F_{k+1} - g_{k+1}^T s_k + \frac{1}{2} s_k^T \nabla^2 F(w_{k+1}) s_k - \frac{1}{6} s_k^T \left(T_{k+1} s_k\right) s_k + O(\|s_k\|^4). \tag{14}$$

In [27], a simple modification of (12) was proposed as $y_k^* = y_k + \text{sign}(\psi_k) \frac{\psi_k}{\|s_k\|^2} s_k$ to handle the case $\psi_k < 0$. We show below that this modification does not provide any improvement.

### 3.1 Sign correction

Considering the equations in (13) together yields

$$\psi_k = \frac{1}{6} s_k^T (T_{k+1}s_k)s_k + O(\|s_k\|^4). \tag{15}$$

Let $\psi_k < 0$. Therefore, we have $s_k^T y_k^* = s_k^T y_k - \psi_k$ which leads to derive

$$\begin{aligned} s_k^T \nabla^2 F(w_{k+1}) s_k - s_k^T y_k^* &= s_k^T \nabla^2 F(w_{k+1}) s_k - \left(s_k^T y_k + \psi_k\right) + 2\psi_k \\ &= \frac{2}{3} s_k^T \left(T_{k+1} s_k\right) s_k + O(\|s_k\|^4). \end{aligned} \tag{16}$$

Equation (16) shows that the dominant error is even worse than the one in (13). Therefore, we suggest to use $y_k$ whenever $\psi_k < 0$; otherwise we can use $y_k^*$.

### 3.2 A new modified secant condition

Taking the derivative of both sides of (14) with respect to $s_k$ and premultiplying it by $s_k^T$ lead to

$$\begin{aligned} s_k^T g_k &= s_k^T g_{k+1} - s_k^T \nabla^2 F(w_{k+1}) s_k + \frac{1}{2} s_k^T \left(T_{k+1} s_k\right) s_k + O(\|s_k\|^4) \\ &= 3\psi_k + s_k^T y_k + O(\|s_k\|^4). \end{aligned} \tag{17}$$

Considering equations (14) and (17) together yields that the third order term disappears and

$$\begin{aligned} s_k^T \nabla^2 F(w_{k+1}) s_k &= 6(F_k - F_{k+1}) + 3 s_k^T \left(g_{k+1} + g_k\right) + s_k^T y_k + O(\|s_k\|^4) \\ &= 3\psi_k + s_k^T y_k + O(\|s_k\|^4), \end{aligned} \tag{18}$$

which suggests the choice of

$$y_k^* = \frac{3\psi_k}{\|s_k\|^2} s_k + y_k. \tag{19}$$

Obviously, the new vector $y_k^*$ in equation (19) provides a better curvature approximation (the error is of order $O(|s_k|^4)$, than the one of order $O(|s_k|^3)$ in equation (12)).

## 4 The modified L-BFGS trust region method

In this section, we define the modified L-BFGS trust region method (M-LBFGS-TR). In solving (1), trust-region methods using either standard or modified $B_k$, generate a sequence of iterates (4) in which $p_k$ is obtained by solving the following trust-region subproblem

$$p_k = \arg\min_{p \in \mathbb{R}^n} Q_k(p) := \frac{1}{2} p^T B_k p + g_k^T p \quad \text{s.t.} \quad \|p\|_2 \leq \delta_k, \tag{20}$$

for some trust-region radius $\delta_k > 0$, where $g_k := \nabla F(w_k)$ and $B_k \approx \nabla^2 F(w_k)$.

The acceptance of the trial (4) is based on the ratio between the actual reduction in the objective function of (1) and the reduction predicted by the quadratic model, that is

$$\rho_k = \frac{F(w_k) - F(w_t)}{Q_k(0) - Q_k(p_k)}. \tag{21}$$

Since the denominator in (21) is nonnegative, if $\rho_k$ is positive, the new iterate $w_{k+1} := w_t$; otherwise, $w_{k+1} := w_k$. The process of adjustment of the trust-region radius at each iteration is described in Algorithm 3.

According to [9,8] the subproblem (20) can be efficiently solved if $B_k$ is chosen to be a quasi-Newton matrix. Let $B_k$ be a (modified) L-BFGS Hessian approximation in compact form (6). As described in [15,28], the global solution of (20) is characterized by the following theorem:

**Theorem 1.** *Let $\delta$ be a given positive constant. A vector $p^*$ is a global solution of the trust region problem* (20) *if and only if $\|p^*\|_2 \leq \delta$ and there exists a unique $\sigma^* \geq 0$ such that $B_k + \sigma^* I$ is positive semi-definite with*

$$(B_k + \sigma^* I)p^* = -g_k, \qquad \sigma^*(\delta_k - \|p^*\|_2) = 0. \tag{22}$$

*Moreover, if $B_k + \sigma^* I$ is positive definite, then the global minimizer is unique.*

Following [1,8,30], the solution of the trust-region subproblem (20) can be computed as

$$p^* := p(\sigma^*) = -\frac{1}{\tau_k} \left( I - \Psi_k \left( \tau_k M_k^{-1} + \Psi_k^T \Psi_k \right)^{-1} \Psi_k^T \right) g_k. \tag{23}$$

**Algorithm 1** Trust-region subproblem solution.

**Inputs:** Current $\Psi \triangleq \Psi_k$, $M^{-1} \triangleq M_k^{-1}$, $\gamma \triangleq \gamma_k$, $\delta \triangleq \delta_k$ and $g \triangleq g_k$
Compute the thin QR factorization of $\Psi$ with factors $Q$ and $R$
Compute the spectral decomposition of matrix $RMR^T = U\hat{\Lambda}U^T$
Set: the reordered matrix $\hat{\Lambda} = \text{diag}(\hat{\lambda}_1, \ldots, \hat{\lambda}_k)$ such that $\hat{\lambda}_1 \leq \ldots \leq \hat{\lambda}_k$
Compute the spectral decomposition of $B_k$ as $\Lambda_1 = \hat{\Lambda} + \gamma I$
Let: $\lambda_{min} = \min\{\lambda_1, \gamma\}$
Compute $P_{\|} = QU$
Compute $g_{\|} = P_{\|}^T g$
Compute $a_j = (g_{\|})_j$ and $a_{k+1} = \sqrt{\|g\|_2^2 - \|g_{\|}\|_2^2}$
**if** $\phi(0) \geq 0$ **then**
    Set: $\sigma^* = 0$
    Compute $p^*$ with (25) as solution of $(B_k + \sigma^* I)p = -g$
**else**
    Compute a root $\sigma^* \in (0, \infty)$ of (27) by Newton's method
    Compute $p^*$ with (25) as solution of $(B_k + \sigma^* I)p = -g$
**end if**

where $\tau_k = \gamma_k + \sigma^*$. This direct formula can be obtained by exploiting the spectral decomposition of the coefficient matrix $B_k + \sigma^* I$ and its inversion using the Sherman-Morrison-Woodbury formula [29].

Algorithm 1 describes the process of solving the trust-region subproblem. It is based on the strategies described in the subsequent paragraphs. For further details see [1,8,30].

The spectral decomposition of matrix $B_k + \sigma^* I$ can be computed as follows. Computing the thin QR factorization of matrix $\Psi$, $\Psi_k = Q_k R_k$, where $Q_k \in \mathbb{R}^{n \times 2k}$ and $R_k \in \mathbb{R}^{2k \times 2k}$, and the cheap spectral decomposition of the $2k \times 2k$ matrix $R_k M_k R_k^T$ as $R_k M_k R_k^T = U_k \hat{\Lambda} U_k^T$, where $U_k$ and $\hat{\Lambda} = \text{diag}(\hat{\lambda}_1, \ldots, \hat{\lambda}_{2k})$ are respectively orthogonal and diagonal matrices, leads to

$$B_k = B_0 + Q_k R_k M_k R_k^T Q_k^T = \gamma_k I + Q_k U_k \hat{\Lambda} U_k^T Q_k^T.$$

Now, let $P_{\|} \triangleq Q_k U_k$ and $P_{\perp} \triangleq (Q_k U_k)^{\perp}$ where $(.)^{\perp}$ denotes orthogonal complement. By Theorem 2.2.1 in [18], we have

$$P^T P = P P^T = I$$

where $P \triangleq \begin{bmatrix} P_{\|} & P_{\perp} \end{bmatrix} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix. Therefore the spectral decomposition of $B_k$ is obtained as

$$B_k = P \Lambda P^T, \qquad \Lambda \triangleq \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}, \tag{24}$$

where

$$\Lambda_1 = \hat{\Lambda} + \gamma_k I = \text{diag}(\hat{\lambda}_1 + \gamma_k, \hat{\lambda}_2 + \gamma_k, \ldots, \hat{\lambda}_{2k} + \gamma_k),$$
$$\Lambda_2 = \gamma_k I.$$

We assume the eigenvalues are increasingly ordered.

*The inversion of* $B_k + \sigma^* I$ Let $\tau_k = \gamma_k + \sigma$. Applying the Sherman-Morrison-Woodbury formula [29] to compute the inverse of the coefficient matrix $B_k + \sigma^* I$ leads to

$$p(\sigma) = -(B_k + \sigma I)^{-1} g_k = -\frac{1}{\tau_k}\left(I - \Psi_k \left(\tau_k M_k^{-1} + \Psi_k^T \Psi_k\right)^{-1} \Psi_k^T\right) g_k. \tag{25}$$

By (24) and (25), we have

$$\|p(\sigma)\| = \sqrt{\left\{\sum_{i=1}^{k} \frac{(g_\parallel)_i^2}{(\lambda_i + \sigma)^2}\right\} + \frac{\|g_\perp\|^2}{(\gamma_k + \sigma)^2}}, \tag{26}$$

where

$$\begin{aligned} g_\parallel &= P_\parallel^T g, \\ \|g_\perp\|^2 &= \|g\|^2 - \|g_\parallel\|^2. \end{aligned}$$

Assume $p_u \triangleq p(0)$ is the solution of the first optimality condition $(B_k + \sigma I)p(\sigma) = -g_k$, for which $\sigma = 0$ makes the second optimality condition $\sigma(\delta_k - \|p(\sigma)\|_2) = 0$. holds. If $\|p_u\| \leq \delta$, using (25) we have $(\sigma^*, p^*) = (0, p_u) = (0, p(0))$. If $\|p_u\| > \delta$, then $p^*$ must lie on the boundary of the trust-region to meake the second optimality condition hold. To impose this, $\sigma^*$ must be the root of the following equation

$$\phi(\sigma) \triangleq \frac{1}{\|p(\sigma)\|} - \frac{1}{\delta} = 0, \tag{27}$$

and can be determined by Newton's method, e.g. the variant proposed in [8]. The global solution of the trust-region subproblem is then $(\sigma^*, p^*) = (\sigma^*, p(\sigma^*))$.

## 5 Stochastic M-LBFGS-TR

In the stochastic setting, the training set is divided into multiple subsets called batches. The process of taking a single batch, computing a subsampled gradient and loss for it and then updating the parameters create one single iteration of a stochastic algorithm. This process is repeated for each batch iteratively until one epoch, that is one pass through all data samples, is completed. After each epoch, the dataset is shuffled and new batches are generated.

Let $J_k$ be a random subset of data at iteration $k$, whose size and index set of the samples included are denoted by $|J_k|$ and $J_k^{idx}$, respectively. In this work, samples are drawn without replacement for batches with fixed size. The subsampled loss and gradient are computed as follows

$$F_k^{J_k} := F^{J_k}(w_k) = \frac{1}{|J_k|} \sum_{i \in J_k^{idx}} f_i(w_k), \qquad g_k^{J_k} := \nabla F^{J_k}(w_k) = \frac{1}{|J_k|} \sum_{i \in J_k^{idx}} \nabla f_i(w_k). \tag{28}$$

In L-BFGS-TR, when the batch $J_k$ changes from one iteration to the next, the updates might be unstable since different data points are used to evaluate the gradient at the beginning (in $w_k$) and at the end of the iteration (in the trial $w_t$), and so the gradient difference employed to update the Hessian approximation is computed as $y_k = g_t^{J_{k+1}} - g_k^{J_k}$. To overcome this problem a remedy suggested in [34] consists in using the same multi-batch $J_k$ for computing $y_k = g_t^{J_k} - g_k^{J_k}$ which requires double function and gradient evaluations at $w_k$ and $w_t$. Another sampling strategy was proposed in [2] to compute $y_k = g_t^{O_k} - g_k^{O_k}$ where $O_k = J_k \cap J_{k+1} \neq \emptyset$ such that the overlap set $O_k$ should not be insignificant. Similarly, in the stochastic M-LBFGS-TR, when $\psi_k > 0$, the modified vector $y_k^*$ is computed as in (19) with $\psi_k = (F_k^{O_k} - F_t^{O_k}) + (g_k^{O_k} + g_t^{O_k})^T s_k$.

In this work, we take a particular variant of this approach referred as *half overlap sampling* where $J_k = O_{k-1} \cup O_k$. With this sampling strategy, the overall loss and gradients in (28) are computed as

$$F_k^{J_k} = \frac{1}{2}(F_k^{O_{k-1}} + F_k^{O_k}), \qquad g_k^{J_k} = \frac{1}{2}(g_k^{O_{k-1}} + g_k^{O_k}). \tag{29}$$

This requires two function and gradient evaluations on the overlap set of the currect multi-batch. The M-LBFGS-TR training algorithm is outlined in Algorithm 2. Besides the previously indicated function and gradient evaluations, which constitute the predominant cost, the per iteration complexity of both L-BFGS-TR and M-LBFGS-TR algorithms consists in $2rn + O(r^3)$ operations needed to update $B_k$, and in the trust-region framework, $2(4r+1)n+O(r^2)$ flops to compute $Q(p)$ needed for $\rho$ evaluation and to obtain the search direction $p(\sigma)$ using the direct formula described in (23). We also have the cost of computing a QR factorization and a cheap eigenvalue decomposition requiring $O(nr^2)$ and $O(r^3)$ operations, respectively.

Computing the numerator in (21) using subsampled function differences as $F_t^{J_k} - F_k^{J_k}$ requires double function evaluation at the beginning and at the end of the iteration. Experimentally, we examined that using overlap $O_k$ in place of $J_k$ provides a more affordable cost per iteration without any detriment in the attainable training accuracy. In support of this statement we have included Figure 5. We note that computing $\psi_k$ in M-LBFGS-TR does not impose any additional cost because it uses subsampled loss and gradient values corresponding to $O_k$ which have been already evaluated in the previous iteration.

## 6 Experiments

We summarize in this section the behaviour of the described quasi-Newton optimization algorithms L-BFGS-TR [30] and M-LBFGS-TR on the training of two deep neural networks with different architectures for image classification of the benchmark datasets MNIST and CIFAR10 (see [24,21]). We used Glorot (Xavier) approach [16] for initializing the learning parameters. The architecture of the networks, which contain batch normalization layers, is described below.

– **LeNet-5.** A well known convolutional neural network designed for handwritten and machine-printed character recognition [25]. By solving an optimization problem for $w \in \mathbb{R}^{431,080}$, LeNet-5 with the following architecture is trained with the MNIST dataset:

  - Input layer with a $28 \times 28 \times 1$ image
  - Convolutional layer with 20 filters of $5 \times 5$ size, stride 1 followed by ReLU
  - Max pooling layer with a $2 \times 2$ and stride 2
  - Convolutional layer with 50 filters of $5 \times 5$ size, stride 1 followed by ReLU
  - Max pooling layer with a $2 \times 2$ and stride 2
  - Fully connected layer with 500 neurons followed by ReLU
  - Fully connected layer with 10 neurons followed by softmax

– **ConvNet3FC2.** Motivated by [31], we define a CNN with 3 intermediate convolutional networks (ConvNet) and 2 fully connected networks (FC). This network with the structure defined below, is trained with CIFAR10 by solving an optimization problem for $w \in \mathbb{R}^{3,525,162}$:

  - Input layer with a $32 \times 32 \times 3$ image by imposing Z-score normalization [1]
  - Convolutional layer with 32 filters of $5 \times 5$ size, stride 1 and padding 2
  - Batch normalization layer followed by ReLU
  - Max pooling layer with a $2 \times 2$ window and stride 1
  - Convolutional layer with 32 filters of $5 \times 5$ size, stride 1 and padding 2
  - Batch normalization layer followed by ReLU
  - Max pooling layer with a $2 \times 2$ window and stride 1
  - Convolutional layer with 64 filters of $5 \times 5$ size, stride 1 and padding 2
  - Batch normalization layer followed by ReLU
  - Max pooling layer with a $2 \times 2$ window and stride 1
  - Fully connected layer with 64 neurons,
  - Batch normalization layer followed by ReLU
  - Fully connected layer with 10 neurons followed by softmax

[1] Subtract the mean specified by mean and divide by the standard deviation.

**Algorithm 2** Stochastic M-LBFGS-TR

**Inputs:** the number of multi-batches in one epoch $\bar{N}$, overlap set size $os$, number of epochs $epoch$, limited memory parameter $r$, $k = 0$, $e_k = 0$, $w_0 \in \mathbb{R}^n$, empty matrices $S_0 = Y_0 = [.]$
**while** $k \geq 0$ **do**
  **if** $k = 0$ **then**
    Take the first subset $O_{-1}$, and compute $F_0^{O_{-1}} \triangleq F^{O_{-1}}(w_0)$ and $g_0^{O_{-1}} \triangleq \nabla F^{O_{-1}}(w_0)$
    Take the second subset $O_0$, and compute $F_0^{O_0} \triangleq F^{O_0}(w_0)$ and $g_0^{O_0} \triangleq \nabla F^{O_0}(w_0)$
    Compute $F_k^{J_0} = \frac{1}{2}(F_k^{O_{-1}} + F_k^{O_0})$ and $g_k^{J_0} = \frac{1}{2}(g_k^{O_{-1}} + g_k^{O_1})$
  **else**
    Take the second subset $O_k$ of multi-batch $J_k$
    Compute $F_k^{O_k} \triangleq F^{O_k}(w_k)$ and $g_k^{O_k} \triangleq \nabla F^{O_k}(w_k)$, and then $F_k^{J_k}$ and $g_k^{J_k}$
    Compute $F_k^{J_k} = \frac{1}{2}(F_k^{O_{k-1}} + F_k^{O_k})$ and $g_k^{J_k} = \frac{1}{2}(g_k^{O_{k-1}} + g_k^{O_k})$
    **if** $\mathrm{mod}(k+1, \bar{N}) = 0$ **then**
      $e_k = e_k + 1$ and shuffle the data
    **end if**
  **end if**
  **if** $\|g_k^{J_k}\| \leq \epsilon_1$ or $e_k = epoch$ **then**
    **return**
  **end if**
  **if** $k = 0$ or $S_k = [.]$ **then**
    Compute $p_k = -\delta g_k^{J_k} / \|g_k^{J_k}\|$
  **else**
    Compute $p_k$ using Algorithm 1
  **end if**
  Compute trial $w_t = w_k + p_k$
  Compute $F_t^{O_k} \triangleq F^{O_k}(w_t)$, $g_t^{O_k} \triangleq \nabla F^{O_k}(w_t)$, and then $y_k = g_t^{O_k} - g_k^{O_k}$ and $s_k = w_t - w_k$
  Compute $\psi_k = (F_k^{O_k} - F_t^{O_k}) + s_k^T(g_k^{O_k} + g_t^{O_k})$
  **if** $\mathrm{sign}(\psi_k) > 0$ **then**
    $y_k = y_k + \frac{3\psi_k}{\|s_k\|^2} s_k$
  **end if**
  Compute $\rho_k = (F_t^{O_k} - F_k^{O_k})/Q(p_k)$
  **if** $\rho_k \geq \tau_1$ **then**
    $w_{k+1} = w_t$
  **else**
    $w_{k+1} = w_k$
  **end if**
  Update: $\delta_k$ with Algorithm 3
  **if** $s_k^T y_k > \epsilon_2 \|s_k\|^2$ **then**
    **if** $k \leq r$ **then**
      Store: $s_k$ and $y_k$ as new column in $S_{k+1}$ and $Y_{k+1}$
    **else**
      Keep: only the $r$ recent $\{s_j, y_j\}_{j=k-l+1}^{k}$ in $S_{k+1}$ and $Y_{k+1}$
    **end if**
    Compute the smallest eigenvalue $\hat{\lambda}$ of the problem $(L_k + D_k + L_k^T)u = \lambda S_k^T S_k u$
    **if** $\hat{\lambda} > 0$ **then**
      $\gamma_{k+1} = \max\{1, 0.9\hat{\lambda}\} \in (0, \hat{\lambda})$
    **else**
      Compute $\gamma_k^h$ as $\frac{y_{k-1}^T y_{k-1}}{y_{k-1}^T s_{k-1}}$ and set $\gamma_{k+1} = \max\{1, \gamma_k^h\}$
    **end if**
    Compute $B_0 = \gamma_{k+1} I$, $\Psi_{k+1}$ and $M_{k+1}^{-1}$
  **else**
    Set $B_0 = \gamma_k I$, $\Psi_{k+1} = \Psi_k$ and $M_{k+1}^{-1} = M_k^{-1}$
  **end if**
  $k = k + 1$
**end while**

All experiments were run on an Ubuntu Linux server virtual machine with 32 CPUs and 128GB RAM using MATLAB and its deep learning toolbox. We provide a comparison with the most popular first-order method Adam implemented using the MATLAB built-in functions `sgdmupdate` and `adamupdate` by a grid search tuning effort on learning rate and batch sizes. The best learning rate for all batch sizes is $10^{-3}$. The limited memory parameter for both quasi-Newton methods was set to $r = 20$. We obtained comparable results using different values of $r \in \{5, 10, 15, 20\}$ but we did not include these results here due to space limitation issues. Other hyperparameters for L-BFGS-TR and M-LBFGS-TR algorithms are $\epsilon_1 = 10^{-5}$, $\epsilon_2 = 10^{-2}$, $\gamma_0 = 1$, $\tau_1 = 10^{-6}$, $\tau_2 = 0.1$, $\tau_3 = 0.75$, $\eta_2 = 0.5$, $\eta_3 = 0.8$, $\eta_4 = 2$.

We have investigated the effect of the batch size on the performance of the different training algorithms. The networks were trained for a maximum number of epochs. The program stops before that limit if 100% accuracy has been reached. Figures 1 and 2 show the evolution of loss and accuracy for different batch sizes $|J_k| \in \{100, 500, 2000, 5000\}$ in the classification of MNIST and CIFAR10, respectively. The results corresponding to the smallest batch size for the MNIST dataset are reported within the first epoch only to facilitate the comparison. All the loss and accuracy evolution curves have been filtered by a fixed display frequency. This frequency, when indicated, corresponds to how many iterations per epoch have not been displayed. We observe from Figures 1 and 2 that, for both problems, both sL-BFGS-TR and sM-LBFGS-TR perform better than *tuned* Adam independently of the batch size. In all the experiments, sM-LBFGS-TR exhibits a comparable performance with respect to sL-BFGS-TR. Neither sL-BFGS-TR nor sM-LBFGS-TR are strongly influenced by batch size. Large multi-batch sizes can be employed without a considerable loss of accuracy even though the performance of both methods decreases when larger batch sizes are used, due to the smaller number of iterations per epoch (smaller number of parameters updates). Adam performs very well in both problems providing comparable accuracies to the ones yielded by second-order methods, even if it is less accurate when large batch sizes are used.

Figure 3 displays the variability of the obtained test accuracy computed over five runs with random seeds. It can be seen that the results are reliable and that first-order methods exhibit larger variability than the two quasi-Newton algorithms. According to the complexity analysis performed in the former section, we found that the training time of both second-order methods is larger than that of the first-order ones and that the measured CPU times are comparable for both algorithms of the same type (see Table 1). Nevertheless, we underline the fact that as Figure 4 illustrates, with a fixed computational time budget the proposed quasi-Newton methods provide comparable or better testing accuracy than the first-order Adam optimizer.

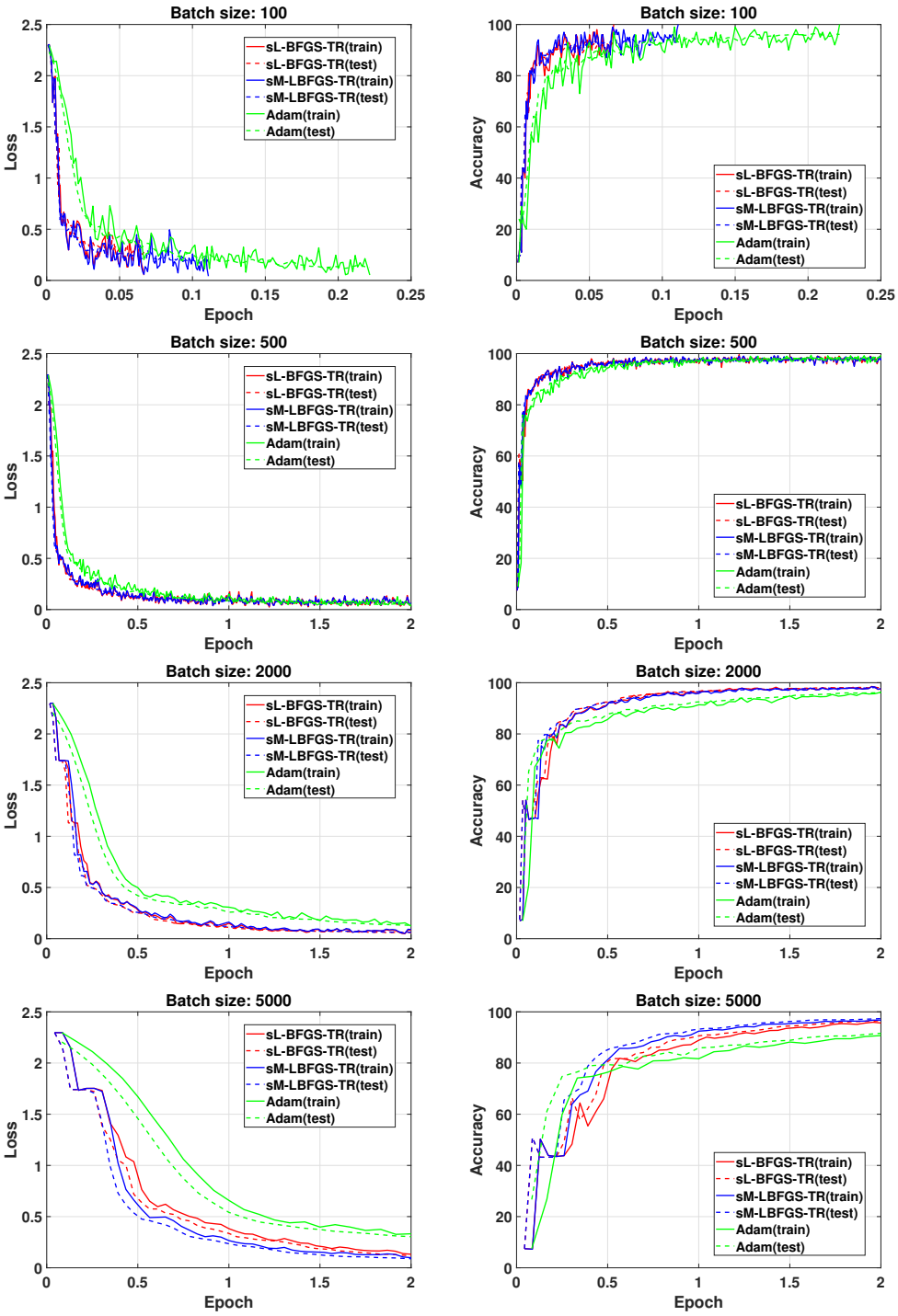


**Fig. 1.** MNIST: Evolution of the training and testing loss and accuracy using stochastic quasi-Newton based methods and *tuned* Adam for different batch sizes.

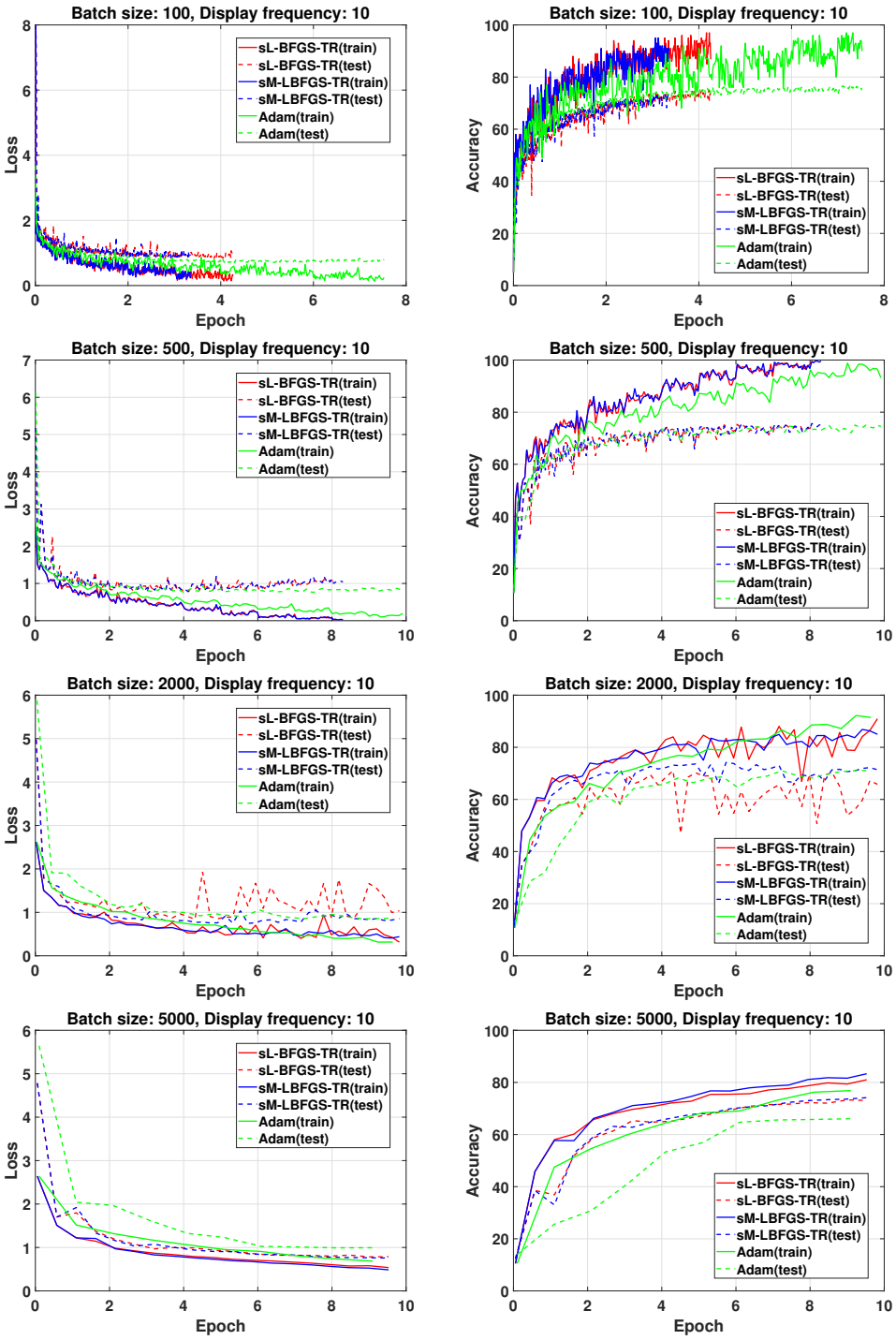


**Fig. 2.** CIFAR10: Evolution of the training and testing loss and accuracy using stochastic quasi-Newton based methods and *tuned* Adam for different batch sizes.

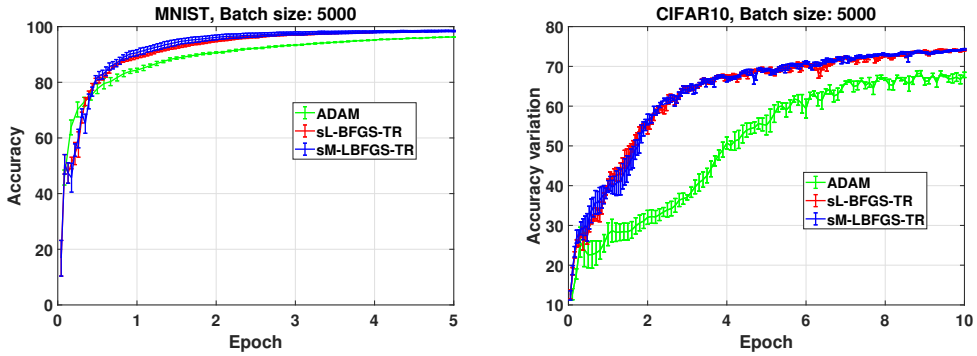


**Fig. 3.** Error bars of stochastic quasi-based methods and *tuned* Adam: variability of the test accuracy in the format "mean±standard deviation" computed over five runs with random seeds.

**Table 1.** Training time of the methods for $k_{max}$ iterations.

| | CIFAR10 ($k_{max} = 100$) | | MNIST ($k_{max} = 200$) | |
|---|---|---|---|---|
| | $bs = 500$ | $bs = 5000$ | $bs = 500$ | $bs = 5000$ |
| SGD | 00:18:37 | 00:41:54 | 00:03:21 | 00:07:26 |
| Adam | 00:18:30 | 00:41:24 | 00:03:30 | 00:07:46 |
| L-BFGS-TR | 00:32:04 | 00:55:04 | 00:09:35 | 00:13:45 |
| M-LBFGS-TR | 00:32:06 | 00:54:46 | 00:09:42 | 00:13:40 |

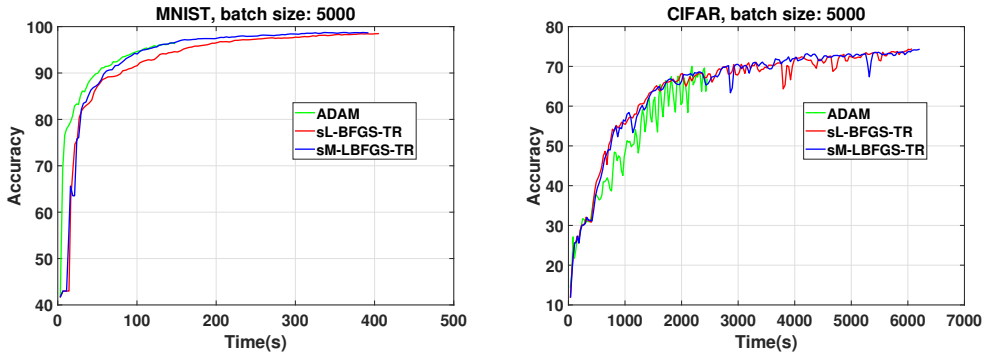


**Fig. 4.** Testing accuracy of stochastic quasi-based methods and *tuned* Adam versus training CPU time (in seconds).

**Algorithm 3**

**Inputs:**

- Current $\delta_k$ and $\rho_k$
- $0 < \tau_2 < 0.5 < \tau_3 < 1$
- $0 < \eta_2 \leq 0.5$
- $0.5 < \eta_3 < 1 < \eta_4$

**if** $\rho_k > \tau_3$ **then**
  **if** $\|p_k\| \leq \eta_3 \delta_k$ **then**
    $\delta_{k+1} = \delta_k$
  **else**
    $\delta_{k+1} = \eta_4 \delta_k$
  **end if**
**else if** $\tau_2 \leq \rho_k \leq \tau_3$ **then**
  $\delta_{k+1} = \delta_k$
**else**
  $\delta_{k+1} = \eta_2 \delta_k$
**end if**

## 7 Conclusions

In this work we have considered stochastic limited memory BFGS quasi-Newton methods using an overlapping sampling strategy for their required computations to solve the nonlinear and non-convex optimization problems arising in the training of deep neural networks. We have provided a comparison of the standard L-BFGS method with a variant of this algorithm based on a modified secant condition which is theoretically shown to provide an increased order of accuracy in the approximation of the curvature of the Hessian. In our experiments, on image

classification problems with MNIST and CIFAR10 datasets, both sL-BFGS-TR and sM-LBFGS-TR exhibit comparable performances. Moreover, figures illustrate that these methods converge faster than *tuned* Adam and perform better for larger batch sizes which are favorable for parallel computing. By considering the largest batch size, our results show that with a fixed computational time budget the proposed quasi-Newton methods provide comparable or better testing accuracy than the first-order Adam optimizer. Nevertheless, despite their better convergence properties and not requiring such a time-consuming tuning effort needed for Adam, the computational complexity per iteration is high. For this reason, future research will be devoted to devising sampling strategies that reduce the number of loss and gradient evaluations per iteration. In addition, the efficiency of these stochastic quasi-Newton based optimizers should be compare with other recently proposed second-order methods such as K-FAC [17]. Another future line of research we are currently undergoing is the analysis of whether better results could be achieved using symmetric rank-one updates and thus allowing for indefinite Hessian approximations.

## References


1. Lasith Adhikari, Omar DeGuchy, Jennifer B Erway, Shelby Lockhart, and Roummel F Marcia. Limited-memory trust-region methods for sparse relaxation. In *Wavelets and Sparsity XVII*, volume 10394. International Society for Optical Engineering, 2017.
2. Albert S Berahas, Jorge Nocedal, and Martin Takáč. A multi-batch L-BFGS method for machine learning. *In Advances in Neural Information Processing Systems*, pages 1055–1063, 2016.
3. Albert S Berahas and Martin Takáč. A robust multi-batch L-BFGS method for machine learning. *Optimization Methods and Software*, 35(1):191–219, 2020.
4. R. Bollapragada, R. Byrd, and J. Nocedal. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis 39 (2), 545-578, 2019.*, 2019.
5. Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, 2018.
6. Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis*, 39(2):545–578, 2019.
7. Léon Bottou and Yann LeCun. Large scale online learning. *Advances in neural information processing systems*, 16:217–224, 2004.
8. Johannes Brust, Jennifer B Erway, and Roummel F Marcia. On solving L-SR1 trust-region subproblems. *Computational Optimization and Applications*, 66(2):245–266, 2017.
9. Oleg Burdakov, Lujin Gong, Spartak Zikrin, and Ya-xiang Yuan. On efficiently combining limited-memory and trust-region techniques. *Mathematical Programming Computation*, 9(1):101–134, 2017.
10. Andrew R Conn, Nicholas IM Gould, and Philippe L Toint. *Trust region methods*. SIAM, 2000.

11. Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, volume 4, pages 2933–2941, 2014.
12. Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.
13. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.
14. Jennifer B Erway, Joshua Griffin, Roummel F Marcia, and Riadh Omheni. Trust-region algorithms for training responses: machine learning methods using indefinite hessian approximations. *Optimization Methods and Software*, 35(3):460–487, 2020.
15. David M Gay. Computing optimal locally constrained steps. *SIAM Journal on Scientific and Statistical Computing*, 2(2):186–197, 1981.
16. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
17. Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. *arXiv preprint arXiv:2006.08877*, 2020.
18. Gene H Golub and Charles F Van Loan. *Matrix computations, 4th Edition*. Johns Hopkins University press, 2013.
19. Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26:315–323, 2013.
20. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.
21. Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Available at: https://www.cs.toronto.edu/~kriz/cifar.html*, 2009.
22. Vyacheslav Kungurtsev and Tomas Pevny. Algorithms for solving optimization problems arising from deep neural net models: smooth problems. *arXiv preprint arXiv:1807.00172*, 2018.
23. Sudhir Kylasa, Fred Roosta, Michael W Mahoney, and Ananth Grama. GPU accelerated sub-sampled newton's method for convex classification problems. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 702–710. SIAM, 2019.
24. Yann LeCun. The MNIST database of handwritten digits. *Available at: http://yann.lecun.com/exdb/mnist/*, 1998.
25. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
26. J. Martens and I. Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.
27. Farzin Modarres, Abu Hassan Malik, and Wah June Leong. Improved hessian approximation with modified secant equations for symmetric rank-one method. *Journal of computational and applied mathematics*, 235(8):2423–2431, 2011.
28. Jorge J Moré and Danny C Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

29. Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
30. Jacob Rafati and Roummel F Marcia. Improving L-BFGS initialization for trust-region methods in deep learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 501–508. IEEE, 2018.
31. Vivek Ramamurthy and Nigel Duffy. L-SR1: A second order optimization method for deep learning. 2016.
32. Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
33. Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.
34. Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *Artificial intelligence and statistics*, pages 436–443. PMLR, 2007.
35. Zengxin Wei, Guoyin Li, and Liqun Qi. New quasi-Newton methods for unconstrained optimization problems. *Applied Mathematics and Computation*, 175(2):1156–1188, 2006.
36. Liu Ziyin, Botao Li, and Masahito Ueda. SGD may never escape saddle points. *arXiv preprint arXiv:2107.11774*, 2021.

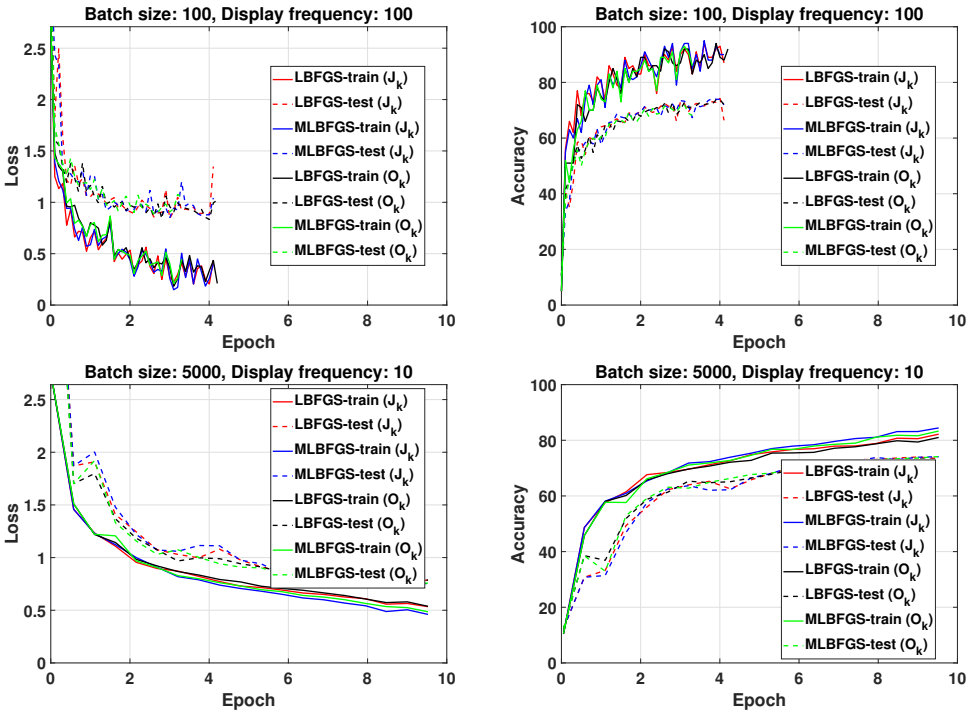


**Fig. 5.** CIFAR10: Evolution of the training and testing loss and accuracy using quasi-Newton methods by using different sample sets ($O_k$ or $J_k$) to compute the difference between the subsampled loss function needed to compute $\rho$.