To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–23

Trust-Region Algorithms for Training Responses: Machine Learning Methods Using Indefinite Hessian Approximations

Jennifer B. Erway[a*], Joshua Griffin[b], Roummel F. Marcia[c], and Riadh Omheni[b]

[a] *Wake Forest University, Winston-Salem, NC;*
[b] *SAS, Cary, NC;* [c] *University of California, Merced, Merced, CA*

(*December, 2017*)

Machine learning (ML) problems are often posed as highly nonlinear and nonconvex unconstrained optimization problems. Methods for solving ML problems based on stochastic gradient descent are easily scaled for very large problems but may involve fine-tuning many hyper-parameters. Quasi-Newton approaches based on the limited-memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) update typically do not require manually tuning hyper-parameters but suffer from approximating a potentially indefinite Hessian with a positive-definite matrix. Hessian-free methods leverage the ability to perform Hessian-vector multiplication without needing the entire Hessian matrix, but each iteration's complexity is significantly greater than quasi-Newton methods. In this paper we propose an alternative approach for solving ML problems based on a quasi-Newton trust-region framework for solving large-scale optimization problems that allow for indefinite Hessian approximations. Numerical experiments on a standard testing data set show that with a fixed computational time budget, the proposed methods achieve better results than the traditional limited-memory BFGS and the Hessian-free methods.

Keywords: Large-scale optimization, non-convex, machine learning, trust-region methods, quasi-Newton methods, limited-memory symmetric rank-one update

AMS Subject Classification: 90C53; 15A06; 90C06; 65K05; 65K10; 49M15

1. Introduction

Machine learning problems, such as text classification and speech recognition, are often nonlinear and nonconvex unconstrained problems of the form

$$\min_{w\in\Re^m} f(w) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i(w), \tag{1}$$

where f_i is a function of the ith observation in a *training* data set $\{(x_i, y_i)\}$ with $x_i \in \Re^d$ and $y_i \in \Re^o$. In the literature (see e.g., [7, 65]), (1) is often referred to as the *empirical risk*. Generally speaking, these problems have several features that make traditional optimization algorithms ineffective. First, both m and n are very large (e.g., typically $m, n \geq 10^6$). Second, there is a special type of redundancy that is present in (1) due to similarity between data points; namely, if $\mathcal{I}$ is a random subset of indices of $\{1, 2, \ldots, n\}$,

*Corresponding author. Email: rmarcia@ucmerced.edu then provided $\mathcal{I}$ is large enough, but $|\mathcal{I}| \ll n$ (e.g., $n = 10^9$ and $|\mathcal{I}| = 10^5$), then

$$\hat{f}(w) \triangleq \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} f_i(w) \approx \frac{1}{n}\sum_{i=1}^{n} f_i(w). \tag{2}$$

The underlying goal during the optimization phase in machine learning is to find the "best" set of model parameters w so that the chosen model function $p(x, w) : \Re^d \times \Re^m \rightarrow \Re^o$ predicts the observed target variable $y \in \Re^o$ as accurately as possible. The most popular approaches in machine learning include (i) stochastic gradient descent method, (ii) limited-memory BFGS, and (iii) Hessian-free methods. Here we briefly describe each approach and describe their advantages and disadvantages.

(i) Stochastic gradient descent (SGD) methods. The stochastic gradient descent (SGD) method [54] is one of the most popular types of methods for solving machine learning problems. For this iterative method, an index j is randomly chosen from $\{1, 2, \ldots n\}$ at each iteration and w is updated as follows:

$$w = w - \eta_j \nabla f_j(w),$$

where ∇f_j denotes the gradient of f_j. The parameter η_j is referred to as the *learning rate* in machine learning. SGD is a very attractive algorithm for machine learning for several reasons. First, it naturally exploits data set redundancy described in (2); moreover, the iteration complexity is independent of n. In contrast, classical optimization algorithms are explicitly dependent on n and become much more unstable when attempting exploit cheaper stochastic approximations of the gradient [7, 12, 18, 19, 31, 48, 58]. Second, the algorithm comes with attractive convergence theory [7]. Third, the SGD algorithm readily responds to an on-line learning environment (i.e., data is available in a sequential order instead of all-at-once) where data observations may never repeat. A fourth advantage occurs in the nonconvex setting where the stochastic nature of SGD makes it much less likely to converge to inferior local minimums [13, 34, 57] than non-stochastic methods.

There are several important disadvantages associated with using SGD. To enhance the performance of SGD in practice, developers must fine-tune many hyper-parameters– leading to many variations of SGD, (e.g., see [1, 23, 33, 35, 63, 67, 68]). One set of hyper-parameter users must choose is a learning rate sequence (i.e., $\{\eta_j\}$). If the learning rate is too small, the algorithm may stall; on the other hand, if the learning rate is too large the algorithm may not converge. In practice, finding an effective sequence $\{\eta_j\}$ can require solving the same problem many times to find the best sequence. This dilemma has led to a resurgence of interest in auto-tune algorithms that can aid the SGD user in this search [2, 4–6, 22, 37, 40, 60, 61]. A second disadvantage with SGD is that it is inherently sequential, making it difficult to parallelize [21, 37, 44, 53].

(ii) Limited-memory BFGS (L-BFGS). One of the most popular classical algorithms in machine learning is the L-BFGS algorithm, which falls into the class of limited memory quasi-Newton algorithms. Like SGD, quasi-Newton algorithms require only first-order (gradient) information. Quasi-Newton methods generate a sequence of iterates using the rule

$$w_{k+1} = w_k + \eta_k p_k, \quad \text{where} \quad p_k \triangleq -B_k^{-1}\nabla f(w_k), \tag{3}$$

B_k is a *quasi-Newton* matrix that is updated at each iteration using gradient information, and η_k is a suitably-defined step length (learning rate). The update to B_k is defined using sequences of vectors $\{s_j\}$ and $\{y_j\}$, which are given as

$$s_j \triangleq w_{j+1} - w_j \qquad \text{and} \qquad y_j \triangleq \nabla f(w_{j+1}) - \nabla f(w_j), \tag{4}$$

for $j = 0, \ldots, k-1$. The Broyden class of updates, parametrized by $\phi \in \Re$, is the most widely-used updating rule for B_k:

$$B_{k+1} = B_k - \frac{1}{s_k^T B_k s_k} B_k s_k s_k^T B_k + \frac{1}{y_k^T s_k} y_k y_k^T + \phi(s_k^T B_k s_k) v_k v_k^T, \tag{5}$$

where

$$v_k = \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}.$$

In practice, B_0 is usually taken to be a positive scalar multiple of the identity. In large-scale optimization, *limited-memory* quasi-Newton methods are used to bound storage requirements and promote efficiency. In this case, only the r most-recently computed pairs $\{(s_j, y_j)\}$ are used to build B_{k+1}, i.e., only the most up-to-date information is used to model the Hessian matrix. The value of r is typically very small so that $r \ll n$.

While the matrices in the sequence $\{B_j\}$ are symmetric by construction, different choices of ϕ lead to sequences of matrices with different properties. The most well-known member of the Broyden class of updates is the *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* update, which is obtained by setting $\phi = 0$. Provided B_0 is positive definite and $y_i^T s_i > 0$ for each $i = 0, \ldots, k-1$, then the BFGS update generates a sequence of symmetric positive-definite matrices. (The condition $y_i^T s_i > 0$ for each i can be enforced using a *Wolfe line search* to compute η_k in (3).) One reason why the BFGS update is the preferred update is that there is an efficient way to solve linear systems with B_k, making the computation of p_k in (3) affordable [49]. It is worth noting that of all the quasi-Newton updates available, the limited-memory BFGS (L-BFGS) update has been used almost exclusively by researchers in machine learning.

L-BFGS has several advantages in the machine learning setting. First, the computation of $\nabla f(w)$ benefits from a parallel-programming environment. Second, while there are only a few hyper-parameters that the user may tune, such as the number of weights (m) used and the scaling for the initial matrix B_0, there are known standard initializations and values used by the optimization community; that is, L-BFGS does not require manual tuning.

L-BFGS has a number of disadvantages for solving problems in machine learning, especially in deep learning, where the network is composed of multiple cascading layers. First, it cannot be used in an on-line learning environment without significant modifications that limit its scalability to arbitrarily large data sets. (This has given rise to recent research into *stochastic* L-BFGS variations that have thus far been unable to maintain the robustness of classical L-BFGS in a stochastic mini-batch environment [7, 12, 18, 19, 31, 48, 58].) A third disadvantage of L-BFGS occurs if one tries to enforce positive definiteness of the L-BFGS matrices in a nonconvex setting. In this case, L-BFGS has the difficult task of approximating an indefinite matrix (the true Hessian) with a positive-definite matrix B_k, which can result in the generation of nearly-singular matrices $\{B_k\}$. Numerically, this creates need for heuristics such as periodically reinitializing B_k to a multiple of the identity, effectively generating a steepest-descent direction in the next iteration. This can be a significant disadvantage for neural network problems where model quality is highly correlated with the quality of initial steps [43].

(iii) Hessian-free (HF) methods. A third family of algorithms of interest come from classical algorithms that can leverage the ability to perform Hessian-vector multiplies without needing the entire Hessian matrix itself [20, 41–43]; for this reason, as in [41, 43], we will refer to this class as *Hessian-free* algorithms. These algorithms perform approximate updates of the form

$$w_{k+1} = w_k + \eta_k p_k \quad \text{with} \quad \nabla^2 f(w_k) p_k = -\nabla f(w_k), \tag{6}$$

where p_k is an approximate Newton direction obtained computed using a *conjugate-gradient*-like (CG-like) algorithm and η_k is the step length. Traditional CG algorithms assume $\nabla^2 f(w_k)$ is positive definite and solve for p_k in (6) using only matrix-vector products, and thus, are applicable in large problems in machine learning. Because $\nabla^2 f(w_k)$ may be indefinite in deep learning problems, modified variants are needed to adapt for local nonconvexity; we refer to such approaches as *modified conjugate-gradient* algorithms (MCG).

Remarkably, Martens [41], was able to show that Hessian-free methods were able to achieve out-of-the-box competitive results compared to manually-tuned SGD on deep learning problems. Moreover, Pearlmutter [52] was able to show that matrix-vector products could be computed at a computational cost on the order of a gradient evaluation. However, since multiple matrix-vector products can be required to solve (6), the iteration complexity of MCG is significantly greater than L-BFGS. Thus, despite its allure of being a tune-free approach to deep learning, Hessian-free methods are for the most part unused and unexplored in practice.

Contributions of the proposed method. While the BFGS update is the most widely-used type of quasi-Newton method for general optimization as well as general machine learning, it enjoys certain benefits (given by guaranteed positive-definite Hessian approximations) that may actually hinder it in solving large nonconvex optimization problems. Our proposed approach is based on a different quasi-Newton update, namely the *symmetric rank-1* (SR1) update, which allows for indefinite Hessian approximation. We use a trust-region framework (see e.g., [17]) because this framework can accommodate indefinite Hessian approximations more easily (see [50]). We also present a stochastic extension of our proposed approach, which improves computational time because it does not compute the full gradient at each iteration.

2. L-SR1 trust-region methods

We begin by discussing the SR1 update and trust-region methods for large-scale optimization.

2.1 *The SR1 update*

The SR1 update is the unique rank-one update in the Broyden class satisfying the so-called *secant condition:*

$$B_{k+1} s_k = y_k.$$

This update occurs by setting $\phi = y_k^T s_k/(y_k^T s_k - s_k^T B_k s_k)$ in (5); in this case,

$$B_{k+1} = B_k + \frac{1}{s_k^T(y_k - B_k s_k)}(y_k - B_k s_k)(y_k - B_k s_k)^T, \tag{7}$$

where s_k and y_k are defined in (4). At each iteration, we assume $(y_k - B_k s_k)^T s_k \neq 0$, i.e., all of the updates are well-defined; the update is skipped otherwise (see [50, Sec. 6.2]). This update has the distinction of being the only rank-one update in the Broyden class of updates. Moreover, this update is self-dual: The recursion (7) can be used to generate B_{k+1}^{-1} by interchanging y_k and s_k everywhere in (7) and initializing with B_0^{-1}. Thus, linear systems with SR1 matrices can be solved efficiently. An important aspect of the SR1 update is that regardless of the sign of $y_i^T s_i$ for each i, this update generates a sequence of matrices that may be indefinite. It is precisely this property of SR1 matrices that makes them attractive in applications like deep learning where f is nonconvex.

Decreasing the index by 1 in (7), the SR1 update can be written recursively and compactly using the outer product representation

$$B_k = B_0 + \begin{bmatrix} \\ \Psi_k \\ \\ \end{bmatrix} \begin{bmatrix} M_k \end{bmatrix} \begin{bmatrix} \quad \Psi_k^T \quad \end{bmatrix} \tag{8}$$

where $B_0 = \gamma I$ for some $\gamma \neq 0$, Ψ_k is an $n \times k$ matrix and M_k is a $k \times k$ matrix. In the literature, (8) is referred to as the *compact formulation* of an SR1 matrix. In particular, Byrd et al. [10] show that for SR1 matrices,

$$\Psi_k = Y_k - B_0 S_k \quad \text{and} \quad M_k = (D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1}. \tag{9}$$

with $S_k \triangleq [\, s_0 \;\; s_1 \;\; s_2 \;\; \cdots \;\; s_{k-1} \,] \in \Re^{n\times k}$, and $Y_k \triangleq [\, y_0 \;\; y_1 \;\; y_2 \;\; \cdots \;\; y_{k-1} \,] \in \Re^{n\times k}$, and L_k is the strictly lower triangular part, U_k is the strictly upper triangular part, and D_k is the diagonal part of $S_k^T Y_k = L_k + D_k + U_k$. In our proposed approach, we use a limited-memory SR1 (L-SR1) update, where only r of the most recent pairs $\{s_j, y_j\}$ are stored, where the value of r is typically very small so that $r \ll n$.

While SR1 updates are one of many updates proven to theoretically converge to the Hessian matrix at a minimizer, there is some evidence that in practice SR1 updates have superior convergence properties [16].

The SR1 advantage: Historically, the SR1 update fell out of favor when it appeared to suffer from more algorithmic breakdowns and instabilities than the BFGS update; however, simple safeguards are now used to adequately prevent instabilities and breakdowns [50, p.145]. Over the last several decades, the SR1 update has reemerged as the subject of much research; in fact, in [29, p.118], Gould states: "[SR1] has now taken its place alongside the BFGS method as the pre-eminent updating formula"

For machine learning, the SR1 update offers distinct advantages over the BFGS update: (i) In machine learning problems, Wolfe line searches to enforce $y_k^T s_k > 0$ in BFGS methods are too computationally expensive to use which has led to the popular solution of skipping BFGS updates, possibly degrading the quality of the Hessian approximation [50, p.146]; (ii) SR1 matrices exhibit better convergence to the true Hessian (e.g., see the discussion on convergence in Section 2.1); and (iii) if one tries to generate a sequence of positive-definite L-BFGS matrices when modeling an indefinite Hessian, the matrices in this sequence may become nearly singular (i.e., highly ill-conditioned) with the smallest eigenvalue of this sequence of matrices becoming close to zero. Since machine learning problems are nonconvex, it is worth noting that (ii) and (iii) suggest that SR1 matrices may generate more accurate approximations than positive-definite L-BFGS matrices of the true Hessian. Moreover, when (iii) occurs, the search direction obtained from a BFGS method may be of poor quality, hindering convergence of the overall method. In fact, research on SR1 methods have produced comparable, if not better, results to BFGS methods [15, 16] on general optimization problems.

2.2 *Large-scale trust-region methods*

Trust-region methods minimize a function f by modeling changes in the objective function using quadratic models. Each iteration requires *approximately* solving a *trust-region subproblem.* Specifically, at the kth iteration, the kth trust-region subproblem is given by

$$p^* \;=\; \underset{p\in\Re^n}{\operatorname{argmin}} \quad \mathcal{Q}_k(p) \triangleq g_k^T p + \frac{1}{2}p^T B_k p \qquad \text{subject to} \quad \|p\| \le \delta_k, \tag{10}$$

where $g_k \triangleq \nabla f(w_k)$, $B_k \approx \nabla^2 f(w_k)$, and δ_k is a given positive *trust-region radius.* Basic trust-region methods update the current approximate minimizer for f only if the ratio between the actual and predicted change in function value is sufficiently large. If the ratio is sufficiently large, the update is accepted and $w_{k+1} \leftarrow w_k + p^*$. When this is not the case, δ_k is reduced and the trust-region subproblem is resolved. The solution of the trust-region subproblem is the computational bottleneck of most trust-region methods. The primary advantage of using a trust-region method is that B_k does not have to be a positive-definite matrix; in particular, it may be a limited-memory SR1 matrix.

Trust-region methods for general large scale optimization use an iterative method to solve the trust-region subproblem. It is well known that when the two-norm is used to define the subproblem (10), we can completely characterize a global solution of the subproblem. The optimality conditions for the trust-region subproblem defined using the two-norm are due to Gay [28] and Moré and Sorensen [47]:

Theorem: *Let δ be a given positive constant. A vector p^* is a global solution of the trust-region problem* (10) *if and only if $\|p^*\|_2 \le \delta$ and there exists a unique $\sigma^* \ge 0$ such that $B_k + \sigma^* I$ is positive semidefinite with*

$$(B_k + \sigma^* I)p^* = -g_k, \quad \text{and} \quad \sigma^*(\delta_k - \|p^*\|_2) = 0. \tag{11}$$

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

Most iterative methods for solving the trust-region subproblem assume it is possible to compute matrix-vector products with the true Hessian, but matrix factorizations are too computationally expensive to perform. Examples of such methods include Steihaug's method [62], Toint's method [64], the GLTR method [30], phased-SSM [25], Hager's SSM method [32], Erway and Gill's SSM method [24], and the LSTRS method [55, 56]. In many machine learning applications, these methods are too computationally expensive for use on the full data set.

2.3 *Solving the L-SR1 trust-region subproblem*

Solving the trust-region subproblem (10) is generally the computational bottleneck of trust-region methods. In recent work by the authors [8], an efficient algorithm for solving the trust-region subproblem (10) is proposed, where B_k is the SR1 quasi-Newton update. To efficiently solve the subproblems, we exploit the structure of the L-SR1 matrix to obtain global solutions to high accuracy. We summarize this approach here.

To begin, we transform the optimality equations (11) using the spectral decomposition of B_k, which we outline here (see [8] for more details). Given the compact formulation of B_k, $B_k = B_0 + \Psi M \Psi^T$, and the "thin" QR factorization of Ψ, $\Psi = QR$, then $B_k = \gamma I + QRMR^TQ^T$, where $B_0 = \gamma I$ and $\gamma > 0$ (see [9, 26]). Since RMR^T is a small $k \times k$ matrix, its spectral decomposition $V\hat{\Lambda}V^T$ can be quickly computed. Then, letting $\Pi \triangleq [\, QV \quad (QV)^{\perp}] \in \Re^{n\times n}$ such that $\Pi^T\Pi = \Pi\,\Pi^T = I$, the spectral decomposition of B_k is given by

$$B_k = \Pi\Lambda\Pi^T, \text{ where } \Lambda \triangleq \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} = \begin{bmatrix} \hat{\Lambda} + \gamma I & 0 \\ 0 & \gamma I \end{bmatrix}, \tag{12}$$

where $\Lambda_1 = \text{diag}(\,\hat{\lambda}_1 + \gamma, \hat{\lambda}_2 + \gamma, \ldots, \hat{\lambda}_k + \gamma) \in \Re^{k\times k}$, and $\Lambda_2 = \gamma I_{n-k}$. Using the spectral decomposition of B_k, the optimality equations (11) become

$$(\Lambda + \sigma^* I)v^* = -\Pi^T g \tag{13a}$$

$$\sigma^*(\delta_k - \|v^*\|_2) = 0, \tag{13b}$$

for some scalar $\sigma^* \geq 0$ and $v^* = \Pi^T p^*$, where p^* is the global solution to (10). The Lagrange multiplier σ^* can be obtained by substituting the expression

$$\|v^*\|_2^2 = \|(\Lambda + \sigma^* I)^{-1}\Pi^T g\|_2^2 = \sum_{i=1}^{k} \frac{(\Pi^T g)_i^2}{(\hat{\lambda}_i + \gamma} + \sigma^*)^2 + \frac{\|\left((QV)^{\perp}\right)^T g\|_2^2}{(\gamma + \sigma^*)^2} \tag{14}$$

from (13a) into (13b) and finding the largest solution σ^* to the secular equation

$$\phi(\sigma) = \frac{1}{\|v(\sigma)\|_2} - \frac{1}{\delta} = 0 \tag{15}$$

using Newton's method. Once σ^* is obtained, v^* can be computed from (13a) and as well as the solution $p^* = \Pi v^*$ to the original trust-region subproblem (10). Note that in only one special case, the so-called *hard case* [17, 46], the above method will not work because the computed $\|p^*\|$ will not lie on the boundary of the trust region. In this case, the global solution to the trust-region subproblem is given by $p^* = \hat{p}^* + \alpha u_{\min}$, where $\hat{p}^* = -(B_k + \sigma^* I)^{\dagger} g$, $u_{\min}$ is a column of P and is an eigenvector associated with the most negative eigenvalue of B_k and can be computed from the partial spectral decomposition outlined above, and $\alpha = \pm\sqrt{\delta^2 - \|\hat{p}^*\|^2}$ is a scalar to ensure that p^* lies on the boundary. (See [8] for details on the hard case.)

2.4 *Proposed approach*

The proposed L-SR1 Trust-Region Method (L-SR1-TR) is outlined in Algorithm 1, and the trust-region subproblem solver is described in Algorithm 2. For details on the subproblem solver and all related computations, see [8, Algorithm 1].

Algorithm 1 L-SR1 Trust-Region (L-SR1-TR) Method

```
Require: $x_0 \in \Re^n$, $\delta_0 > 0$, $\epsilon > 0$, $\gamma_0 > 0$ , $0 \le \tau_1 < \tau_2 < 0.5 < \tau_3 < 1$,
    $0 < \eta_1 < \eta_2 \le 0.5 < \eta_3 < 1 < \eta_4$, $\alpha = 1$
1: Compute $g_0$
2: for $k = 0, 1, 2, \ldots$ do
3:   if $\|g_k\| \le \epsilon$ then
4:     return
5:   end if
6:   Choose at most $m$ pairs $\{s_j, y_j\}$
7:   Compute $p^*$ using Algorithm 2
8:   Compute step-size $\alpha$ with Wolfe line-search on $p^*$. Set $p^* = \alpha p^*$.
9:   Compute the ratio $\rho_k = (f(w_k + p^*) - f(w_k))/\mathcal{Q}_k(p^*)$
10:  $w_{k+1} = w_k + p^*$
11:  Compute $g_{k+1}$, $s_k$, $y_k$, and $\gamma_k$
12:  if $\rho_k < \tau_2$ then
13:    $\delta_{k+1} = \min(\eta_1\delta_k, \eta_2\|s_k\|_2)$
14:  else
15:    if $\rho_k \ge \tau_3$ and $\|s_k\|_2 \ge \eta_3\delta_k$ then
16:      $\delta_{k+1} = \eta_4\delta_k$
17:    else
18:      $\delta_{k+1} = \delta_k$
19:    end if
20:  end if
21: end for
```

2.5 *Stochastic extension*

In this section, we describe how to improve the efficiency of L-SR1-TR by incorporating approximate gradient calculations derived from random sampling of the training data. The use of mini-batches can be motivated by considering (2), which suggests a (potentially significantly smaller) subset may be sufficient to obtain a meaningful descent direction for the true objective function. Mini-batching refers to the process whereby a subset of training data is used to approximate the full gradient calculation each iteration. That is, instead of using g_k the gradient is approximated by

$$\tilde{g}_k \triangleq \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(w_k) \approx \nabla f(w_k), \tag{16}$$

where $\mathcal{I}_k \subseteq \{1, 2, \ldots, n\}$. Obviously as $|\mathcal{I}_k|$ decreases the savings in computational cost must be weighed against the resulting degradation in progress. Remarkably first-order algorithms like SGD function behave quite well even if $\mathcal{I}_k$ consists of only a single observation at each iteration. The reason is that the gradient error can be shown to cancel itself out in the expected value sense. However, for higher-order approaches such as quasi-Newton methods, the batch size typically needs to be larger. Further, batch sizes need not be fixed–strategies for dynamically increasing batch size have been studied in [11, 45, 59]. In our experience, we have found robustness in starting with an arbitrarily small batch

Algorithm 2 Orthonormal Basis SR1 Method

```
1: Compute the Cholesky factor $R$ of $\Psi^T\Psi$;
2: Compute the spectral decomposition $RMR^T = U\hat{\Lambda}U^T$ (with $\hat{\lambda}_1 \leq \cdots \leq \hat{\lambda}_k$);
3: Let $\Lambda_1 = \hat{\Lambda} + \gamma I$;
4: Let $\lambda_{\min} = \min\{\lambda_1, \gamma\}$, and let $r$ be its algebraic multiplicity;
5: Define $g_{\|} \triangleq (\Psi R^{-1} U)^T g$;
6: if $\lambda_{\min} > 0$ and $\phi(0) \geq 0$ [the unconstrained minimizer is feasible] then
7:    $\sigma^* = 0$ and compute $p^* = -B_k^{-1} g_k$;
8: else if $\lambda_{\min} \leq 0$ and $\phi(-\lambda_{\min}) \geq 0$ then
9:    $\sigma^* = -\lambda_{\min}$;
10:   Solve $(B_k + \sigma^* I)p^* = -g_k$;
11:   if $\lambda_{\min} < 0$ [the hard case] then
12:      Compute $\alpha^*$ and $u^*_{\min}$;
13:      $p^* \leftarrow p^* + \alpha^* u^*_{\min}$;
14:   end if
15: else
16:   Use Newton's method to find $\sigma^*$, a root of $\phi$, in $(\max\{-\lambda_{\min}, 0\}, \infty)$;
17:   Solve $(B_k + \sigma^* I)p^* = -g_k$;
18: end if
```

size and increasing the batch size whenever progress towards the minimizer appears to stagnate.

For this work, we use overlapping training samples [3], requiring that at each iteration the mini-batch $\mathcal{I}_k$ is formed using a prescribed percentage of overlap with the previous mini-batch. That is, at the kth iteration, the overlap $\mathcal{I}_k \cap \mathcal{I}_{k-1}$ is predetermined. Using overlapping mini-batches and (16), the quasi-Newton pairs $\{(s_{k-1}, y_{k-1})\}$ are computed as

$$s_{k-1} = w_k - w_{k-1} \quad \text{and} \quad y_{k-1} = \tilde{g}_k - \tilde{g}_{k-1}.$$

As with SGD, there is inherent noise in the search direction due to using (16) instead of the true gradient. A common approach to mitigate the effects of this noise is to use the principles of momentum, which is the exponential averaging of recent steps. Specifically, in our approach we add the following momentum term at the end of each iteration:

$$v_k = \mu v_{k-1} + (w_k - w_{k-1}).$$

The most commonly-used value for the momentum parameter is $\mu = .9$ (see e.g., [63]). The momentum step v_k is grafted into the trust-region solution p^* from (10) as follows:

$$v_k \leftarrow \mu \min\left(1.0, \frac{\delta_k}{\|v_k\|}\right) v_k, \tag{17a}$$

$$p^* \leftarrow \min\left(1.0, \frac{\delta_k}{\|p^* + v_k\|}\right)(p^* + v_k), \tag{17b}$$

where δ_k denotes the current trust-region radius (see Algorithm 1). Note that if $\mu = 0$, then the trust-region step p^* would be left unchanged by the above transformation. We call this approach Limited-Memory Stochastic SR1 Trust-Region (L-SSR1-TR), and it differs from Alg. 1 (L-SR1-TR) in three specific places: Line 1, which uses the approximate gradient $\tilde{g}_0$ instead the exact initial gradient g_0; Line 7, which incorporates the momentum step v_k into the trust-region subproblem solution p^*; and Line 11, which uses the approximate gradient $\tilde{g}_{k+1}$ instead the exact initial gradient g_{k+1} and would compute y_k using the approximate gradient, i.e., $y_k = \tilde{g}_{k+1} - \tilde{g}_k$. L-SSR1-TR is outlined in Algorithm 3.

L-SSR1-TR requires the use of two new hyper-parameters (the momentum parameter and the mini-batch overlap parameter). Unlike SGD where convergence is very sensitive to the learning rate, we have found that convergence of the proposed method is not adversely affected by small changes in these hyper-parameters. In fact, we have found that these parameters are no more sensitive to tuning than the existing quasi-Newton parameters such as memory size and the trust-region expansion and contraction parameters (see η_1 and η_2 in Algorithm 1).

Algorithm 3 Limited-Memory Stochastic SR1 Trust-Region (L-SSR1-TR) Method

```
Require: $x_0 \in \Re^n$, $\delta_0 > 0$, $\epsilon > 0$, $\gamma_0 > 0$, $0 \le \tau_1 < \tau_2 < 0.5 < \tau_3 < 1$,
    $0 < \eta_1 < \eta_2 \le 0.5 < \eta_3 < 1 < \eta_4$, $\alpha = 1$, $\mu = 0.9$
 1: Compute initial batch $\mathcal{I}_0$ and $\hat{g}_0$
 2: for $k = 0, 1, 2, \ldots$ do
 3:   if $\|\hat{g}_k\| \le \epsilon$ then
 4:     return
 5:   end if
 6:   Choose at most $m$ pairs $\{s_j, y_j\}$
 7:   Compute $p^*$ using Algorithm 2
 8:   $v_k = \mu v_{k-1} + (w_k - w_{k-1})$
 9:   $v_k = \mu \min(1.0, \delta_k / \|v_k\|) v_k$
10:   $p^* = \min(1.0, \delta_k / \|p^* + v_k\|)(p^* + v_k)$
11:   Compute step-size $\alpha$ with Wolfe line-search on $p^*$. Set $p^* = \alpha p^*$.
12:   Compute the ratio $\rho_k = (\hat{f}(w_k + p^*) - \hat{f}(w_k))/\mathcal{Q}_k(p^*)$
13:   $w_{k+1} = w_k + p^*$
14:   Compute $\hat{g}_{k+1}$, $s_k$, $y_k$, and $\gamma_k$
15:   if $\rho_k < \tau_2$ then
16:     $\delta_{k+1} = \min(\eta_1 \delta_k, \eta_2 \|s_k\|_2)$
17:   else
18:     if $\rho_k \ge \tau_3$ and $\|s_k\|_2 \ge \eta_3 \delta_k$ then
19:       $\delta_{k+1} = \eta_4 \delta_k$
20:     else
21:       $\delta_{k+1} = \delta_k$
22:     end if
23:   end if
24:   Compute batch $\mathcal{I}_{k+1}$ in accordance with Assumption 4
25: end for
```

2.5.1 Line-search analysis

Here, we demonstrate that under some mild assumptions, the line-search step in Algorithm 3 is guaranteed to a step length that sufficiently decreases $\hat{f}(w)$. We first state these assumptions.

Assumption 1 Let the mini-batch set of observations $\mathcal{I}_k$ be sampled randomly with

$n_b = |\mathcal{I}_k|$. Then there exists a positive function $\gamma : \mathbb{R} \to \mathbb{R}$ such that:

$$\left\| \left\{ \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(w) \right\} - \nabla f(w) \right\|_\infty \leq \gamma(n_b) \tag{18}$$

where $\gamma(n_b) \to 0$ as $n_b \to n$.

This assumption suggests that as $|\mathcal{I}_k|$ increases, $\tilde{g}_k$ in (16) approaches $\nabla f(w)$.

Assumption 2 The line search in Algorithm 3 is performed only on the sampled function $\hat{f}(w)$.

This assumption requires that the line search uses the same batch that was used to define the trust-region subproblem.

Assumption 3 Algorithm 3 negates the search direction whenever $\hat{g}(w)^T p^* > 0$, where p^* is from (17b).

Next, we make the following assumption to ensure that we are making progress in decreasing the full empirical risk $f(w)$.

Assumption 4 The objective f is fully evaluated every $J > 1$ iterations (say, at iterates $w_{J_0}, w_{J_1}, w_{J_2}, \ldots$, where $0 \leq J_0 < J$ and $J = J_1 - J_0 = J_2 - J_1 = \cdots$) and nowhere else in the algorithm. The batch size n_b is monotonically increased whenever

$$f(w_{J_\ell}) > f(w_{J_{\ell-1}}) - \tau$$

for some $\tau > 0$.

This assumption states that if progress is not made in decreasing f, the batch size is increased to reduce the noise associated with using a subsampled surrogate function $\hat{f}$.

Given Assumptions 1 through 4, we now present convergence results for L-SR1-TR. The theorem below asserts the trust-region radius update will always succeed.

Theorem 2.1 *At iteration k, given the batch $\mathcal{I}_k$, the line-search step in Algorithm 3 can never fail. That is, there exists $\alpha > 0$ such that the strong-Wolfe conditions hold:*

(1) $\hat{f}(w_k + \alpha p^*) \leq \hat{f}(w_k) + c_1 \alpha \nabla \hat{f}(w_k)$
(2) $|\nabla \hat{f}(w_k + \alpha p^*)^T p^*| \leq c_2 |\nabla \hat{f}(w_k)^T p^*|$.

Proof. Because each $f_i(w)$ is smooth, the function $\hat{f}(w)$ is likewise smooth. Thus because the search direction is a descent direction for $\hat{f}(w)$, the result follows. Because of Assumption 2 and smoothness assumptions on elements $f_i(w)$, classical line-search proofs hold so long as the batch $\mathcal{I}_k$ is held constant and not resampled during this stage. ■

Theorem 2.2 *If the momentum parameter $\mu \to 0$, then either*

$$\liminf_{k \to \infty} \|\nabla f(w_k)\| = 0 \quad \textit{or} \quad \liminf_{k \to \infty} f(w_k) = -\infty. \tag{19}$$

Proof. For simplicity of notation, we will define $\hat{w}_i = w_{J_i}$. By Assumption 4, the objective function must monotonically reduce over the subsequence $\{\hat{w}_i\}$ or $n_b \to n$. Suppose the objective function is decreased ι_k times over the subsequence $\{\hat{w}_i\}_{i=0}^{k}$. Then

$$f(\hat{w}_k) = f(\hat{w}_0) + \sum_{i=1}^{\iota_k} (f(\hat{w}_i) - f(\hat{w}_{i-1})) \leq f(\hat{w}_0) - \iota_k \tau.$$

Assuming $n_b \nrightarrow n$, then as $k \to \infty$, $\iota_k \to \infty$, and (19) holds. If $n_b \to n$, we reduce to a classic line-search approach whose convergence is assured via the trust-region algorithm that makes sufficient progress at each iteration (see e.g., [51]). ■

2.6 *Initial matrix B_0*

In this section we borrow terminology defined in Section 2.3. For simplicity in this section we will assume that $B_0 = \gamma I$ and analyze the impact of γ on various scenarios. We will show in this section that the choice of γ plays a critical role in a trust-region approach. We start by proving a brief lemma summarizing how directions of negative curvature present in B_k affects the trust-region solution, a variation of which may also be found in [66]. We will denote the smallest and largest eigenvalues of B_k by $\lambda_{\min}(B_k)$ and $\lambda_{\max}(B_k)$, respectively.

LEMMA 2.3 *If $B_k \not\succeq 0$, as the trust-region radius δ_k increases, the trust-region solution, p^*, asymptotically becomes parallel to the eigenspace corresponding to $\lambda_{\min}(B_k)$. That is,*

$$\lim_{\delta_k \to \infty} \frac{|u_{\min}^T p^*|}{\|u_{\min}\| \|p^*\|} = 1.$$

where $u_{\min}$ is an eigenvector corresponding to $\lambda_{\min}(B_k)$.

Proof. Without loss of generality, we assume that the $\lambda_{\min}(B_k)$ has multiplicity one for ease of presentation. (For how to handle the general case, the notation in [8] can be used.) Let $\lambda_{\min}(B_k) = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$, and let π_i be the ith column of Π in the eigendecomposition of B_k in (12). Using this notation, $u_{\min} = \pi_1$. Then we can define

$$\|p(\sigma)\|^2 = \|(B_k + \sigma I)^{-1} g_k\|^2 = \|\Pi(\Lambda + \sigma I)^{-1} \Pi^T g_k\|^2 = \sum_{i=1}^{n} \frac{(\pi_i^T g_k)^2}{(\lambda_i + \sigma)^2}, \tag{20}$$

provided $\sigma \neq -\lambda_i$ for $1 \leq i \leq n$. To prove the lemma, we consider two cases: (i) $\pi_1^T g_k \neq 0$ and (ii) $\pi_1^T g_k = 0$.

Case (i): If $\pi_1^T g_k \neq 0$, then rearranging (20) yields

$$\frac{1}{(\lambda_{\min}(B_k) + \sigma)^2} = \frac{1}{(u_{\min}^T g_k)^2} \left(\|p(\sigma)\|^2 - \sum_{i=2}^{n} \frac{(\pi_i^T g_k)^2}{(\lambda_i + \sigma)^2} \right),$$

since $\lambda_{\min}(B_k) = \lambda_1$ and $u_{\min} = \pi_1$. Moreover, as $\sigma \to -\lambda_{\min}(B_k)^+$, then $\|p(\sigma)\| \to \infty$ and $\lim_{\sigma \to -\lambda_{\min}(B_k)^+} \phi(\sigma) = -1/\delta$ (see (15)). Since $\phi(\sigma^*) = 0$ and $\phi(\sigma)$ is continuous on the interval $(-\lambda_{\min}(B_k), \infty)$, the optimal Lagrange multiplier σ^* satisfies $\sigma^* > -\lambda_{\min}(B_k)$

(see Fig. 2.6(a)). Thus, the solution $p^* = -(B_k + \sigma^* I)^{-1} g_k$ satisfies

$$u_{\min}^T p^* = -u_{\min}^T (B_k + \sigma^* I)^{-1} g_k = -\frac{u_{\min}^T g_k}{\lambda_{\min} + \sigma^*}.$$

At the optimal Lagrange multiplier σ^*, the trust-region subproblem solution $p(\sigma^*)$ lies on the boundary, i.e., $\|p(\sigma^*)\| = \delta_k$ and since $\|u_{\min}\| = \|\pi_1\| = 1$, we have

$$\begin{aligned} \lim_{\delta_k \to \infty} \frac{|u_{\min}^T p^*|}{\|u_{\min}\| \|p^*\|} &= \lim_{\delta_k \to \infty} \frac{|u_{\min}^T g_k|}{(\lambda_{\min} + \sigma^*)\delta_k} \\ &= \lim_{\delta_k \to \infty} \left(\delta_k^2 - \sum_{i=2}^{n} \frac{(\pi_i^T g_k)^2}{(\lambda_i + \sigma^*)^2} \right)^{\frac{1}{2}} \cdot \frac{1}{\delta_k} \\ &= 1. \end{aligned}$$

Case (ii): Suppose $\pi_1^T g_k = 0$. For any $\sigma^* \geq -\lambda_{\min}$, the vector $\hat{p}^*$ given by

$$\hat{p}^* = -(B_k + \sigma^*)^{\dagger} g_k = -\sum_{i=2}^{n} \frac{\pi_i^T g_k}{\lambda_i + \sigma^*} \pi_i,$$

satisfies the first optimality condition $(B_k + \sigma^*)\hat{p}^* = -g_k$. Now the length of $\hat{p}^*$ is bounded since $\sigma^* \geq -\lambda_1 > -\lambda_i$ for all $i \geq 2$. Thus, for sufficiently large δ_k, $\|\hat{p}^*\| < \delta_k$, and the trust-region subproblem solution is given by

$$p^* = \hat{p}^* + \alpha u_{\min},$$

where α is chosen such that $\|p^*\| = \delta_k$ (see Sec. 2.3). (Note that this is precisely the hard case (see Fig. 2.6(b).) Since $u_{\min}^T \hat{p}^* = 0$ (see [8]),

$$\lim_{\delta_k \to \infty} \frac{|u_{\min}^T p^*|}{\|u_{\min}\| \|p^*\|} = \lim_{\delta_k \to \infty} \frac{|\alpha|}{\|p^*\|} = \lim_{\delta_k \to \infty} \frac{\sqrt{\delta_k^2 - \|\hat{p}^*\|^2}}{\delta_k} = 1,$$

which completes the proof. ■

Lemma 2.3 shows the importance for B_k to capture curvature information correctly since the trust-region subproblem solution, p^*, becomes more parallel to the eigenvector corresponding to the most negative eigenvalue of B_k. We next prove conditions that highlight how the choice of γ affects B_k.

LEMMA 2.4 *Suppose $B_0 = \gamma I$ and that $\hat{\lambda}$ denotes the smallest eigenvalue of the generalized eigenvalue problem*

$$(D_k + L_k + L_k^T)u = \hat{\lambda} S_k^T S_k u.$$

Further assume that Ψ_k and S_k are full rank. Then if $\hat{\lambda} > 0$, we have the following properties:

(1) B_k is positive definite if $0 < \gamma < \hat{\lambda}$.
(2) As $\gamma \to \hat{\lambda}$ from below, $\lambda_{\max}(B_k) \to \infty$ and $\operatorname{cond}(B_k) \to \infty$.

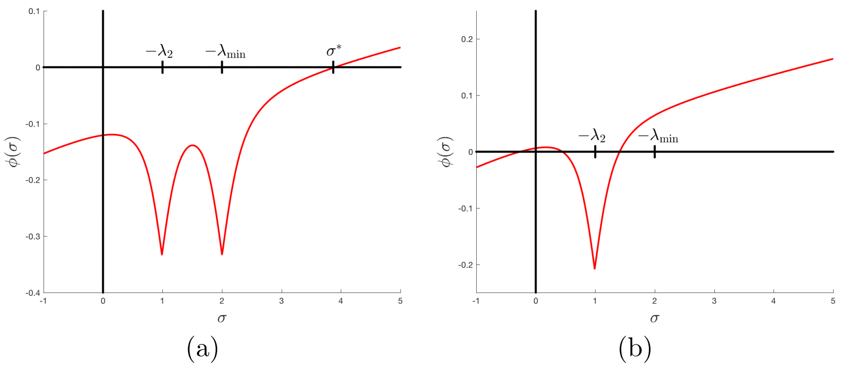


Figure 1. Graph of $\phi(\sigma)$ when $B_k \not\succeq 0$. (a) When $\pi_1^T g \neq 0$, the optimal Lagrange multiplier satisfies $\sigma^* > -\lambda_{\min}(B_k)$. (b) As $\delta_k \to \infty$, $\phi(\sigma)$ behaves as in the hard case: If $\pi_1^T g_k = 0$ and $\lim_{\sigma \to -\lambda_{\min}(B_k)^+} \phi(\sigma) > 0$, then $\hat{p}^* = -(B_k + \sigma^*)^\dagger g_k$ has length $\|\hat{p}^*\| < \delta_k$, where $\sigma^* = -\lambda_{\min}$. In this case, $p^* = \hat{p}^* + \alpha u_{\min}$, where $u_{\min}$ is an eigenvector of B_k corresponding to $\lambda_{\min}(B_k)$ and α is chosen so that $\|p^*\| = \delta_k$.

(3) As $\gamma \to \hat{\lambda}$ from above, $\lambda_{\min}(B_k) \to -\infty$.

Proof. Recall from (8) and (9) that $B_k = \gamma I + \Psi_k M_k \Psi_k^T$, where $\Psi_k = Y_k - \gamma S_k$ and $M_k^{-1} = D_k + L_k + L_k^T - \gamma S_k^T S_k$. Note if $0 < \gamma < \hat{\lambda}$, then $M_k^{-1} \succ 0$, and consequently, $M_k \succ 0$. Hence $B_k \succ 0$. By assumption we have

$$M_k^{-1} u = (\hat{\lambda} - \gamma) S_k^T S_k u.$$

Forming the QR factorization of $S_k = Q_k R_k$ and letting $z = R_k u$, we have that $S_k^T S_k = R_k^T R_k$ and $R_k^{-T} M_k^{-1} R_k^{-1} z = (\hat{\lambda} - \gamma) z$. Consequently,

$$R_k M_k R_k^T z = \frac{1}{(\hat{\lambda} - \gamma)} z.$$

Let $\hat{z}$ be the min-two norm solution to $\Psi_k^T \hat{z} = z$. Then we have that

$$\hat{z}^T B_k \hat{z} = \gamma \|\hat{z}\|^2 + \frac{1}{\hat{\lambda} - \gamma} \|z\|.^2$$

The results then follow since $\hat{z}$ and z are constant and nonzero as γ changes. ■

Lemma 2.4 shows that not choosing γ judiciously in relation to $\hat{\lambda}$ can have deleterious effects. In particular, if γ is too close to $\hat{\gamma}$ from below, then B_k becomes ill-conditioned. If γ is too close to $\hat{\gamma}$ from above, then the smallest eigenvalue of B_k becomes negatively large arbitrarily.

Next, we analyze the relationship between the choice of γ and the conditions under which we can expect $\hat{\lambda} > 0$. First, we note that the predicted reduction $\mathcal{Q}_k(p^*) = \nabla f(w_k)^T p^* + \frac{1}{2} p^{*T} B_k p^*$ is always less for a descent direction p^* if p^* is also a direction of negative curvature. Moreover, from Lemma 2.3, we see that p^* tends to be parallel to the eigenvector of B_k corresponding to its smallest eigenvalue. However, it is desirable to avoid the situation where p^* is a *false* direction of negative curvature, meaning $p^{*T} \nabla^2 f(w_k) p^* > 0$ while $p^{*T} B_k p^* < 0$. The following lemma shows that in the limit, we can select γ so that $0 < \gamma < \hat{\lambda}$, i.e., in the limit, $\hat{\lambda} > 0$ unless the true underlying Hessian is either indefinite or singular.

Lemma 2.5 *Suppose that f is twice-continuously differentiable, that the matrix S_k remains full-rank, and that $w_k \to w^*$, where $\nabla^2 f(w^*) \succ 0$. Then $\hat{\lambda}$ corresponding to B_k is positive in the limit.*

Proof. We observe that each (s_j, y_j) pair satisfy $y_j = \nabla f(w_j + s_j) - \nabla f(x_j)$. Using Taylor expansion, we have that

$$\begin{aligned}
\nabla f(w^* - w^* + w_j + s_j) &= \nabla f(w^*) + \nabla^2 f(w^*)(w_j - w^* + s_j) + t_{j+1} \\
\nabla f(w^* - w^* + w_j) &= \nabla f(w^*) + \nabla^2 f(w^*)(w_j - w^*) + t_j,
\end{aligned}$$

where the components of t_{j+1} and t_j are $\mathrm{O}(\|w_j - w^* + s_j\|^2)$ and $\mathrm{O}(\|w_j - w^*\|^2)$, respectively. Combining these two equations yields

$$y_j = \nabla^2 f(w^*) s_j + (t_{j+1} - t_j). \tag{21}$$

We must prove that there exists a $K > 0$ and $\beta > 0$ such that for all $k > K$,

$$\hat{\lambda} = \min_v \frac{v^T (L_k + D_k + L_k^T) v}{v^T S_k^T S_k v} > \beta.$$

For simplicity let us define $A_k = L_k + D_k + L_k^T$ such that

$$(A_k)_{i+1,j+1} = \begin{cases} s_i^T y_j & \text{for } i \geq j \\ s_j^T y_i & \text{otherwise.} \end{cases}$$

Note from (21) we have that

$$s_i^T y_j = s_i^T \nabla^2 f(w^*) s_j + s_i^T (t_{j+1} - t_j),$$

and thus

$$\begin{aligned}
v^T (L_k + D_k + L_k^T) v &= v^T S_k^T \nabla^2 f(w^*) S_k v + \\
& \quad 2 \sum_{j=k-r}^{k-1} \sum_{i=j}^{k-1} s_i^T (t_{j+1} - t_j) v_{i+1} v_{j+1} + \sum_{j=k-r}^{k-1} s_j^T (t_{j+1} - t_j) v_{j+1}^2.
\end{aligned}$$

Since $s_j = w_{j+1} - w_j$, as w_j converges to w^*, both t_{j+1} and t_j tend to 0. Thus

$$\lim_{k\to\infty} \frac{v^T (L_k + D_k + L_k^T) v}{v^T S_k^T S_k v} = \lim_{k\to\infty} \frac{v^T S_k^T \nabla^2 f(w^*) S_k v}{v^T S_k^T S_k v} \geq \lambda_{\min}(\nabla^2 f(w^*)) > 0$$

by assumption. ∎

In the next lemma, we show that selecting $\gamma > \hat{\lambda}$ can result in a false curvature prediction. To simplify the proof we show that the result holds for a quadratic function. A more general proof simply uses Taylor expansions and asymptotic limit properties.

Lemma 2.6 *Suppose we apply Algorithm 1 to a quadratic objective function $f(w) = c^T w + \frac{1}{2} w^T H w$, where $c \in \Re^n$ and $H \in \Re^{n \times n}$ are both constant. Then if $\gamma = \tau \hat{\lambda}$ with*

$0 < \tau < 1$ *then* B_k *can be indefinite only if the true Hessian is indefinite in the range of* S_k*, that is,*

$$S_k^T \nabla^2 f(w) S_k \not\succeq 0.$$

Conversely, if $\tau > 1$ *then* B_k *may have arbitrarily large negative eigenvalues even if the objective is convex. Furthermore, for any trust-region radius* $\delta_k > 0$,

$$\lim_{\tau \to 1^+} \mathcal{Q}_k(p^*) = -\infty.$$

Thus the model's quality measured by the ratio of actual reduction versus predicted reduction

$$\rho_k = \frac{f(w_k + p^*) - f(w_k)}{\mathcal{Q}_k(p^*).}$$

may be arbitrarily poor for any δ_k *sufficiently large.*

Proof. Note that for a quadratic function $f(w)$,

$$y_k = \nabla f(w_{k+1}) - \nabla f(w_k) = Hw_{k+1} - Hw_k = Hs_k,$$

and therefore, $Y_k = HS_k$. This implies that $S_k^T Y_k = S_k^T H S_k$, and therefore, $L_k + D_k + L_k^T = S_k^T H S_k$. Then from (8) and (9), we have

$$B_k = \gamma I + (H - \gamma I) S_k (S_k^T H S_k - \gamma S_k^T S_k)^{-1} S_k^T (H - \gamma I).$$

If $S_k^T H S_k \succ 0$, then $L_k + D_k + L_k^T \succ 0$ and $\hat{\lambda} > 0$. Thus, if $\gamma = \tau \hat{\lambda}$ with $0 < \tau < 1$, then B_k is positive definite since $(S_k^T H S_k - \gamma S_k^T S_k)$ is positive definite because $0 < \gamma < \hat{\lambda}$.

Conversely, if $\tau > 1$, from the smallest eigenvalue of $(S_k^T H S_k - \gamma S_k^T S_k)$ is negative. Then as $\tau \to 1^+$, $\lambda_{\min}(S_k^T H S_k - \gamma S_k^T S_k)$ approaches 0^-, implying $\lambda_{\min}(B_k)$ approaches $-\infty$. Let $\pi_{\min}$ denote a vector in the eigenspace corresponding to $\lambda_{\min}(B_k)$, scaled so that $\|\pi_{\min}\| = \delta_k$. Then

$$\mathcal{Q}_k(p^*) \le \mathcal{Q}_k(\pi_{\min}) = c^T \pi_{\min} + \frac{1}{2} \pi_{\min}^T B_k \pi_{\min} \le \|c\| \delta_k + \frac{1}{2} \lambda_{\min}(B_k) \delta_k^2. \tag{22}$$

Thus $\lim_{\tau \to 1^+} \mathcal{Q}_k(p^*) = -\infty$. Moreover, for τ sufficiently close to 1 from above, B_k is indefinite, i.e., $\lambda_{\min}(B_k) < 0$, and therefore $\lim_{\delta_k \to \infty} \mathcal{Q}_k(p^*) = -\infty$ in (22). In contrast, if the quadratic objective function is convex, then we must have

$$\lim_{\delta_k \to \infty} f(w_k + p^*) - f(w_k) = \lim_{\delta_k \to \infty} c^T p^* + \frac{1}{2} (p^*)^T H p^* = \infty,$$

meaning that for sufficiently large δ_k, the model function $\mathcal{Q}_k$ poorly predicts the actual reduction in f. ■

When combined with Lemma 2.4, the following lemma suggests selecting a γ near but strictly less than $\hat{\lambda}$ to avoid asymptotically poor conditioning while improving the negative curvature approximation properties of B_k. Note that $\hat{\lambda}$ is cheaply determined due to the small column dimension of S_k.

Lemma 2.7 *Suppose we apply Algorithm 1 to a quadratic objective function $f(w) = c^T w + \frac{1}{2} w^T H w$, where $c \in \Re^n$ and $H \in \Re^{n \times n}$ are constant. Let $\hat{\lambda}$ denote the smallest eigenvalue of the generalized eigenvalue problem*

$$(D_k + L_k + L_k^T)u = \hat{\lambda} S_k^T S_k u.$$

Then for all $\gamma < \hat{\lambda}$, the smallest eigenvalue of B_k is bounded above by the smallest eigenvalue of $\nabla^2 f(w) = H$ in the span of S_k, i.e.,

$$\lambda_{\min}(B_k) \le \min_{S_k v \neq 0} \frac{v^T S_k^T H S_k v}{v^T S_k^T S_k v}.$$

Proof. If $\gamma < \hat{\lambda}$, then the matrix $D_k + L_k + L_k^T - \gamma S_k^T S_k$ is positive definite from the definition of $\hat{\lambda}$. Therefore, from (8) and (12) , the eigenvalues in the matrix $\hat{\Lambda}$ for the low-rank update to B_0 are positive. Consequently, $\lambda_{\min}(B_k) = \gamma$. From the proof of Lemma 2.6, $L_k + D_k + L_k^T = S_k^T H S_k$. Therefore,

$$\hat{\lambda} = \min_{S_k v \neq 0} \frac{v^T S_k^T H S_k v}{v^T S_k^T S_k v}.$$

The result follows from the assumption that $\gamma < \hat{\lambda}$. ■

It is further worthwhile to note that these observations were motivated by investigating why the algorithm failed on some test cases but not others. Once these safe-guards were put in place, the robustness of the algorithm went from inferior to L-BFGS to superior. That is, if the reader has attempted to use L-SR1 in the past and found sometimes it works great, and other times it fails, we suggest that it is likely the case that failures were induced by inadvertently permitting the case $0 < \hat{\lambda} < \gamma$ to occur.

3. Numerical results

In this section, we present two sets of numerical results comparing the performance of several methods, including the proposed Limited-Memory SR1 Trust-Region (L-SR1-TR) and Limited-Memory Stochastic SR1 Trust-Region (L-SSR1-TR) methods, on two databases. In our experiments, we use a fully-connected network model (see Figure 2). The training inputs x_i are 28×28 images, which are represented as vectors in $\Re^{28^2} = \Re^{784}$. At each layer, an affine transformation $W_\ell a_{\ell-1} + b_\ell$ is applied to the input vector $a_{\ell-1}$, where W_ℓ is a matrix of weights and b_ℓ is a bias vector. Before passing onto the next layer, an activation function θ, defined to be the logistic function

$$\theta\left((W_\ell a_{\ell-1} + b_\ell)\right)_j = \frac{1}{1 + e^{-(W_\ell a_{\ell-1} + b_\ell)_j}},$$

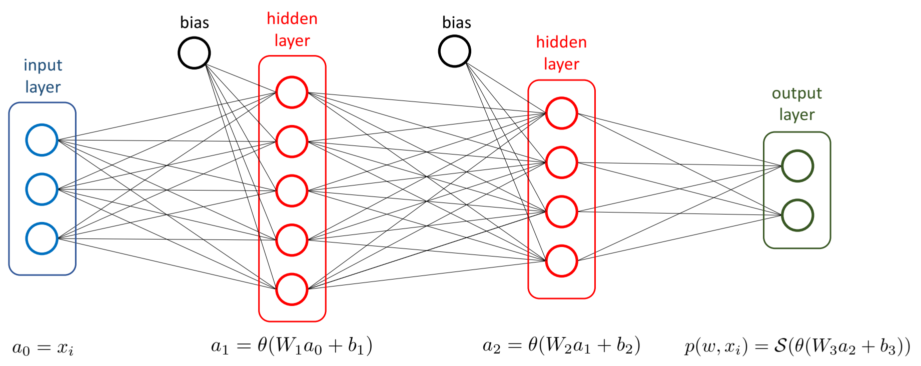

bias
hidden layer
bias
hidden layer
input layer
output layer
$a_0 = x_i$
$a_1 = \theta(W_1 a_0 + b_1)$
$a_2 = \theta(W_2 a_1 + b_2)$
$p(w, x_i) = \mathcal{S}(\theta(W_3 a_2 + b_3))$

Figure 2. Illustration of a fully-connected network model. Here, the input vector is $x_i \in \Re^d$ and the final output vector is $p(w, x_i) \in \Re^K$, with $d = 3$ and $K = 2$. The weight matrices are $W_1 \in \Re^{5\times 3}$, $W_2 \in \Re^{4\times 5}$, and $W_3 \in \Re^{2\times 4}$. The bias vectors are $b_1 \in \Re^5$, $b_2 \in \Re^4$, and $b_3 \in \Re^2$. The activation function $\theta(\,\cdot\,)$ is applied before passing the output to the next layer, except at the input layer. The output vector a_ℓ of hidden layer ℓ is then used as the input in the next layer, $\ell + 1$. In the final (output) layer, the softmax function $\mathcal{S}(\,\cdot\,)$ is applied so that the output vector $p(w, x_i)$ corresponds to probabilities with $\sum_{k=1}^K (p(w, x_i))_k = 1$. Here, $w = (W_1, b_1, W_2, b_2, W_3, b_3)$. When vectorized, $w \in \Re^{54}$.

is applied. At the final layer, L, we apply a softmax function, given by

$$(\mathcal{S}(\theta(W_L a_{L-1} + b_L)))_j = \frac{e^{(\theta(W_L a_{L-1}+b_L))_j}}{\sum_{k=1}^K e^{(\theta(W_L a_{L-1}+b_L))_k}},$$

so that the output vector $p(w, x_i)$ corresponds to probabilities with $\sum_{k=1}^K (p(w, x_i))_k = 1$. Here, $w = (W_1, b_1, W_2, b_2, \ldots, W_L, b_L)$. The softmax function is paired with cross-entropy for the final output layer to form the resulting loss function element f_i in (1):

$$f_i(w) = -\sum_{k=1}^K (y_i)_k \log(p(w, x_i))_k,$$

where K is the dimension of the output layer. For further details, see [27, Chap. 11]. Finally, for L-SSR1-TR, we used 33% for the overlap and used a minibatch size of 100, increasing the batch size by a factor of 1.5 when progress ceased relative to the true loss.

Two errors are used to train a network: training error and test error. The training error is used to define the optimization problem (1). Most approaches that use *training data* tend to find models that *overfit* the data, i.e., the models find relationships specific to the training data that are not true in general. In other words, overfitting prevents machine learning algorithms from correctly generalizing. To help prevent overfitting, an independent data set, called the *test set* is used to validate the accuracy of the model to gage its usefulness in making future predictions. Training errors and test errors are computed using the loss function $f(w)$ in (1). For machine learning, it is important to make sure the trained model yields as small test error as possible. The solution of (1) is taken to be the w that minimizes the test error even though we are directly minimizing the training error, which is our best measure for estimating the expected value of the loss function for unknown data. Generally speaking, with neural network models it is possible to drive the training error to zero for sufficiently large networks; however, the resulting models tend to be overfitted and have less predictive value.

Experiment I. For the first set of experiments, we compared the training and test errors of three methods: (i) a Hessian-free utilizing the Generalized Gauss-Newton method described in [41], (ii) an L-BFGS method based on [39], and (iii) the proposed L-SR1-TR method (see Figure 3). We do not include existing SGD methods because they are already finely tuned for the MNIST data set and the computational time involved in the hyper-parameter tuning cannot easily be accounted for in a fair comparison. For both Ł-BFGS and L-SR1-TR methods, a Wolfe line search was used. We tested the three methods on two data sets with full training and testing observations. The first set (Experiment IA) uses the full Mixed National Institute of Standards and Technology (MNIST) database, which is a large collection of handwritten digits that is commonly used for training various image processing systems [36, 38]. It contains 60,000 training images and 10,000 testing images. The goal is to train the neural network in order to classify the handwritten digits 0 through 9 with minimal error. The second set (Experiment IB) uses the Extended MNIST (EMNIST) database, which is an extension of the MNIST database to handwritten letters [14]. We compared the performance of the three methods on different network configurations with varying numbers of layers and neurons, which are denoted by the sequence of numbers above each graph in Figures 3 and 4. For example, the sequence "784-350-250-150-10" in Figure 3(a) refers to the following: the number of inputs is $784 = 28^2$, which corresponds to the pixel value of the input images, which are 28×28 in size; the number of layers is 3 with 350 neurons in the first layer, 250 in the second, and 150 in the third; and the number of outputs is 10 for the 10 different classes that correspond to the digits from 0 to 9. All tests were performed in MATLAB (R2016b) on a 64-bit 2.67Ghz Intel® Xeon ® CPU E7-8837 machine with 4 processors and 256 GB RAM. These experiments were designed to test the hypothesis that one of the primary reasons why Hessian-free methods outperform BFGS variants in deep learning optimization problems is that they better approximate and exploit negative curvature.

The results on the four different network configurations are given in Figure 3. In Figure 3, loss versus "iterations" and "time" are plotted. Generally speaking, the Hessian-free method outperforms both L-BFGS and L-SR1-TR in terms of achieving the smallest test loss (and training loss) in the fewest iterations, with L-SR1-TR outperforming L-BFGS. However, the cost per iteration for Hessian-free is significantly higher since Hessian-free uses matrix multiplies whereas the quasi-Newton methods use a (much cheaper) single gradient evaluation. Thus, in terms of wall-time, L-SR1-TR is the fastest method, obtaining the best solution in the least amount of time given a one-hour window to solve the given network.

Experiment II. The second set of experiments compares the two proposed L-SR1-TR and the stochastic mini-batch version of L-SR1-TR (L-SSR1-TR) methods on the same network configurations as in the first set of experiments (see Figures 5 and 6). While the L-SR1-TR method achieves lower test and training losses than L-SSR1-TR per iteration (see Figures 5(a,c,e,g) and 6(a,c,e)), L-SSR1-TR is the fastest method in terms of wall-time (see Figures 5(b,d,f,h) and 6(b,d,f)) because the computational cost per iteration for L-SSR1-TR is significantly cheaper.

We note that the results in Figures 3-6 are representative of other experiments.

4. Conclusions

In this paper, we presented an alternative approach for solving machine learning problems that is based on the L-SR1 update that allows for indefinite Hessian approximations. This approach is particularly suitable for non-convex problems where exploiting directions of negative curvature is crucial. Numerical experiments suggest that the proposed approaches (the limited-memory SR1 trust-region and the limited-memory stochastic SR1 trust-region methods) can outperform the more commonly used quasi-Newton approach (L-BFGS) both in terms of computational efficiency and test and training loss.

Acknowledgments

We would like to thank Wenwen Zhou and Alireza Yektamaram for stimulating conversations concerning stochastic quasi-Newton methods with SR1. Further Alireza was instrumental in setting up the stochastic framework used to generate numerical results. J. Erway's research work was funded by NSF Grants CMMI-1334042 and IIS-1741264. R. Marcia's research work was funded by NSF Grants CMMI-1333326 and IIS-1741490.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I.J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D.G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P.A. Tucker, V. Vanhoucke, V. Vasudevan, F.B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, CoRR abs/1603.04467 (2016), Available at `http://arxiv.org/abs/1603.04467`.

[2] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 437–478.

[3] A.S. Berahas, J. Nocedal, and M. Takác, *A multi-batch L-BFGS method for machine learning*, CoRR abs/1605.06049 (2016), Available at `http://arxiv.org/abs/1605.06049`.

[4] J. Bergstra and Y. Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research 13 (2012), pp. 281–305, Available at `http://dl.acm.org/citation.cfm?id=2188395`.

[5] J. Bergstra, D. Yamins, and D.D. Cox, *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures*, in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, Available at `http://jmlr.org/proceedings/papers/v28/bergstra13.html`, 2013, pp. 115–123.

[6] J.S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, *Algorithms for hyper-parameter optimization*, in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, eds., Curran Associates, Inc., 2011, pp. 2546–2554, Available at `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

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

[8] J. Brust, J.B. Erway, and R.F. Marcia, *On solving L-SR1 trust-region subproblems*, Computational Optimization and Applications 66 (2017), pp. 245–266.

[9] O. Burdakov, L. Gong, S. Zikrin, and Y.X. Yuan, *On efficiently combining limited-memory and trust-region techniques*, Mathematical Programming Computation (2016), pp. 1–34, Available at `http://dx.doi.org/10.1007/s12532-016-0109-7`.

[10] R.H. Byrd, J. Nocedal, and R.B. Schnabel, *Representations of quasi-Newton matrices and their use in limited-memory methods*, Math. Program. 63 (1994), pp. 129–156.

[11] R.H. Byrd, G.M. Chin, J. Nocedal, and Y. Wu, *Sample size selection in optimization methods for machine learning*, Math. Program. 134 (2012), pp. 127–155, Available at `http://dx.doi.org/10.1007/s10107-012-0572-5`.

[12] R.H. Byrd, S.L. Hansen, J. Nocedal, and Y. Singer, *A stochastic quasi-Newton method for large-scale optimization*, SIAM Journal on Optimization 26 (2016), pp. 1008–1031, Available at `http://dx.doi.org/10.1137/140954362`.

[13] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, and Y. LeCun, *The loss surface of multilayer networks*, CoRR abs/1412.0233 (2014), Available at `http://arxiv.org/abs/1412.0233`.

[14] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, *EMNIST: an extension of MNIST to handwritten letters*, arXiv preprint arXiv:1702.05373 (2017).

[15] A.R. Conn, N.I. Gould, and P.L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Mathematics of computation 50 (1988), pp. 399–430.

[16] A.R. Conn, N.I.M. Gould, and P.L. Toint, *Convergence of quasi-newton matrices generated by the symmetric rank one update*, Math. Program. 50 (1991), pp. 177–195.

[17] A.R. Conn, N.I.M. Gould, and P.L. Toint, *Trust-Region Methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[18] F. Curtis, *A Self-Correcting Variable-Metric Algorithm for Stochastic Optimization*, in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 632–641.

[19] F.E. Curtis and X. Que, *A quasi-Newton algorithm for nonconvex, nonsmooth optimization with global convergence guarantees*, Mathematical Programming Computation 7 (2015), pp. 399–428, Available at `https://doi.org/10.1007/s12532-015-0086-2`.

[20] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 2933–2941.

[21] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A.W. Senior, P.A. Tucker, K. Yang, and A.Y. Ng, *Large Scale Distributed Deep Networks*, in *Advances in Neural Information Processing Systems 25*, Available at `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks`, 2012, pp. 1232–1240.

[22] I. Dewancker, M. McCourt, S. Clark, P. Hayes, A. Johnson, and G. Ke, *A stratified analysis of Bayesian optimization methods*, CoRR abs/1603.09441 (2016), Available at `http://arxiv.org/abs/1603.09441`.

[23] J.C. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research 12 (2011), pp. 2121–2159, Available at `http://dl.acm.org/citation.cfm?id=2021068`.

[24] J.B. Erway and P.E. Gill, *A subspace minimization method for the trust-region step*, SIAM Journal on Optimization 20 (2009), pp. 1439–1461, Available at `http://link.aip.org/link/?SJE/20/1439/1`.

[25] J.B. Erway, P.E. Gill, and J.D. Griffin, *Iterative methods for finding a trust-region step*, SIAM J. Optim. 20 (2009), pp. 1110–1131, Available at `http://dx.doi.org/10.1137/070708494`.

[26] J.B. Erway and R.F. Marcia, *On efficiently computing the eigenvalues of limited-memory quasi-newton matrices*, SIAM Journal on Matrix Analysis and Applications 36 (2015), pp. 1338–1359.

[27] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, Vol. 1, Springer series in statistics New York, 2001.

[28] D.M. Gay, *Computing optimal locally constrained steps*, SIAM J. Sci. Statist. Comput. 2 (1981), pp. 186–197.

[29] N. Gould, *An introduction to algorithms for continuous optimization*, Oxford University Computing Laboratory Notes, 2006.

[30] N.I.M. Gould, S. Lucidi, M. Roma, and P.L. Toint, *Solving the trust-region subproblem using the Lanczos method*, SIAM J. Optim. 9 (1999), pp. 504–525.

[31] R. Gower, D. Goldfarb, and P. Richtarik, *Stochastic Block BFGS: Squeezing More Curvature out of Data*, in *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 48, 20–22 Jun, Available at `http://proceedings.mlr.press/v48/gower16.html`, PMLR, New York, New York, USA, 2016, pp. 1869–1878.

[32] W.W. Hager, *Minimizing a quadratic over a sphere*, SIAM J. Optim. 12 (2001), pp. 188–208.

[33] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, Available at `http://jmlr.org/proceedings/papers/v37/ioffe15.html`, 2015, pp. 448–456.

[34] K. Kawaguchi, *Deep learning without poor local minima*, in *Advances in Neural Information Processing Systems 29*, D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, and R. Garnett, eds., Curran Associates, Inc., 2016, pp. 586–594, Available at `http://papers.nips.cc/paper/6112-deep-learning-without-poor-local-minima.pdf`.

[35] D.P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, CoRR abs/1412.6980 (2014), Available at `http://arxiv.org/abs/1412.6980`.

[36] E. Kussul and T. Baidyk, *Improved method of handwritten digit recognition tested on mnist database*, Image and Vision Computing 22 (2004), pp. 971–981.

[37] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng, *On optimization methods for*

deep learning, in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, Bellevue, Washington, USA, June, ACM, New York, NY, USA, 2011, pp. 265–272.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE 86 (1998), pp. 2278–2324.

[39] D.C. Liu and J. Nocedal, *On the limited memory method for large scale optimization*, Mathematical Programming B 45 (1989), pp. 503–528.

[40] D. Maclaurin, D.K. Duvenaud, and R.P. Adams, *Gradient-based Hyperparameter Optimization through Reversible Learning*, in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, JMLR Workshop and Conference Proceedings, Vol. 37, Available at `http://jmlr.org/proceedings/papers/v37/maclaurin15.html`, JMLR.org, 2015, pp. 2113–2122.

[41] J. Martens, *Deep learning via Hessian-free optimization*, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 735–742.

[42] J. Martens and I. Sutskever, *Learning Recurrent Neural Networks with Hessian-Free Optimization*, in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, 2011, pp. 1033–1040.

[43] J. Martens and I. Sutskever, *Training deep and recurrent networks with hessian-free optimization*, in *Neural Networks: Tricks of the Trade*, Springer, 2012, pp. 479–535.

[44] H.B. McMahan and M.J. Streeter, *Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning*, in *Advances in Neural Information Processing Systems 27*, Available at `http://papers.nips.cc/paper/5242-delay-tolerant-algorithms-for-asynchronous-distributed-online-learning`, 2014, pp. 2915–2923.

[45] M.R. Metel, *Mini-batch stochastic gradient descent with dynamic sample sizes*, ArXiv e-prints (2017).

[46] J.J. Moré and D.C. Sorensen, *Computing a trust region step*, SIAM J. Sci. and Statist. Comput. 4 (1983), pp. 553–572.

[47] J.J. Moré and D.C. Sorensen, *Newton's method*, in *Studies in Mathematics, Volume 24. Studies in Numerical Analysis*, Math. Assoc. America, Washington, DC, 1984, pp. 29–82.

[48] P. Moritz, R. Nishihara, and M. Jordan, *A Linearly-Convergent Stochastic L-BFGS Algorithm*, in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 51, 09–11 May, Available at `http://proceedings.mlr.press/v51/moritz16.html`, PMLR, Cadiz, Spain, 2016, pp. 249–258.

[49] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comput. 35 (1980), pp. 773–782.

[50] J. Nocedal and S.J. Wright, *Numerical Optimization*, 2nd ed., Springer, New York, 2006.

[51] J. Nocedal and Y.x. Yuan, *Combining trust region and line search techniques*, in *Advances in nonlinear programming*, Springer, 1998, pp. 153–175.

[52] B.A. Pearlmutter, *Fast exact multiplication by the Hessian*, Neural computation 6 (1994), pp. 147–160.

[53] B. Recht, C. Re, S. Wright, and F. Niu, *Hogwild: A lock-free approach to parallelizing stochastic gradient descent*, in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, eds., Curran Associates, Inc., 2011, pp. 693–701, Available at `http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf`.

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

[55] M. Rojas, S.A. Santos, and D.C. Sorensen, *A new matrix-free algorithm for the large-scale trust-region subproblem*, SIAM Journal on Optimization 11 (2001), pp. 611–646.

[56] M. Rojas, S.A. Santos, and D.C. Sorensen, *Algorithm 873: Lstrs: Matlab software for large-scale trust-region subproblems and regularization*, ACM Trans. Math. Softw. 34 (2008), pp. 11:1–11:28, Available at `http://doi.acm.org/10.1145/1326548.1326553`.

[57] L. Sagun, V.U. Güney, and Y. LeCun, *Explorations on high dimensional landscapes*, CoRR abs/1412.6615 (2014), Available at `http://arxiv.org/abs/1412.6615`.

[58] N.N. Schraudolph, J. Yu, and S. Günter, *A Stochastic Quasi-Newton Method for Online Convex Optimization*, in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 2, 21–24 Mar, Available at `http://proceedings.mlr.press/v2/schraudolph07a.html`, PMLR, 2007, pp. 436–443.

[59] S.L. Smith, P.J. Kindermans, and Q.V. Le, *Don't Decay the Learning Rate, Increase the Batch Size*, ArXiv e-prints (2017).

[60] J. Snoek, H. Larochelle, and R.P. Adams, *Practical Bayesian Optimization of Machine Learning Al-*

gorithms, in *Advances in Neural Information Processing Systems 25:*, Available at `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms`, 2012, pp. 2960–2968.

[61] E.R. Sparks, A. Talwalkar, M.J. Franklin, M.I. Jordan, and T. Kraska, *Tupaq: An efficient planner for large-scale predictive analytic queries*, CoRR abs/1502.00068 (2015), Available at `http://arxiv.org/abs/1502.00068`.

[62] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal. 20 (1983), pp. 626–637.

[63] I. Sutskever, J. Martens, G.E. Dahl, and G.E. Hinton, *On the importance of initialization and momentum in deep learning*, in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, Available at `http://jmlr.org/proceedings/papers/v28/sutskever13.html`, 2013, pp. 1139–1147.

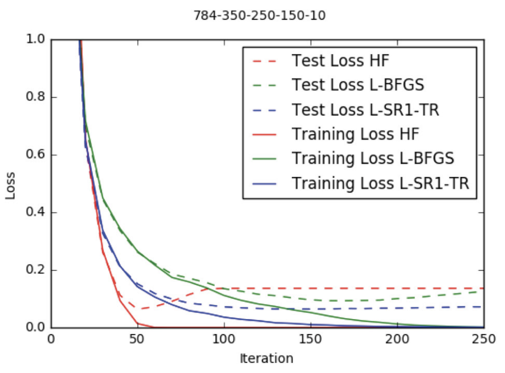
[64] P.L. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, in *Sparse Matrices and Their Uses*, Academic Press, London and New York, 1981, pp. 57–88.

[65] V. Vapnik, *Principles of risk minimization for learning theory*, in *Advances in Neural Information Processing Systems*, 1992, pp. 831–838.

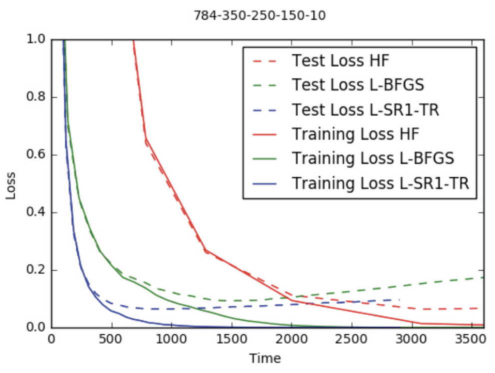
[66] S. Yektamaram, *Optimization algorithms for machine learning designed for parallel and distributed environments*, Ph.D. diss., ISE Department, Lehigh University, Bethlehem, PA, 2017.

[67] M.D. Zeiler, *ADADELTA: an adaptive learning rate method*, CoRR abs/1212.5701 (2012), Available at `http://arxiv.org/abs/1212.5701`.

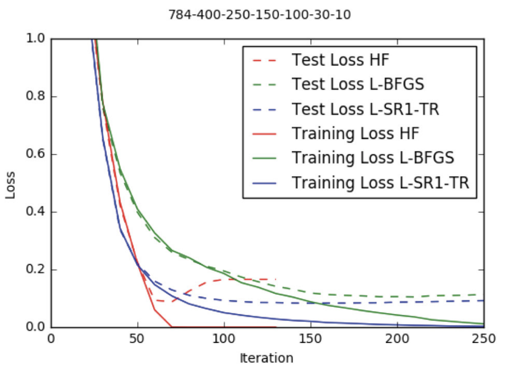
[68] S. Zhang, A. Choromanska, and Y. LeCun, *Deep learning with Elastic Averaging SGD*, in *Advances in Neural Information Processing Systems 28*, Available at `http://papers.nips.cc/paper/5761-deep-learning-with-elastic-averaging-sgd`, 2015, pp. 685–693.



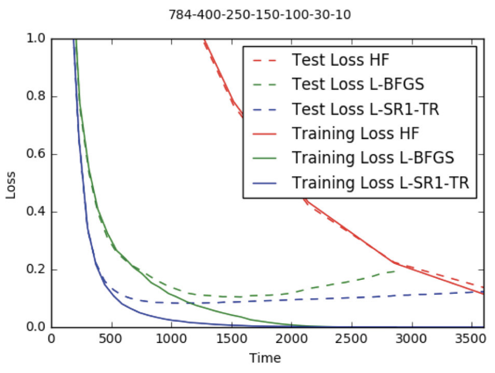
(a)



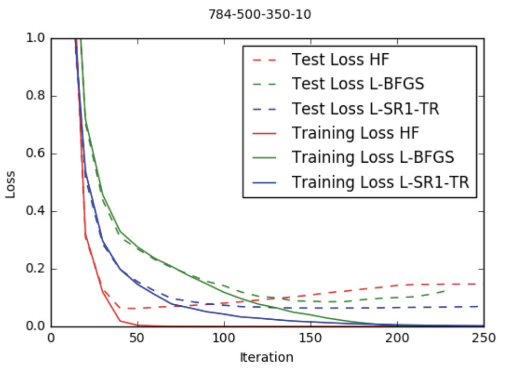
(b)



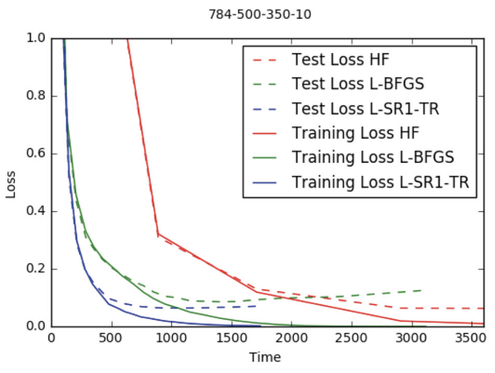
(c)



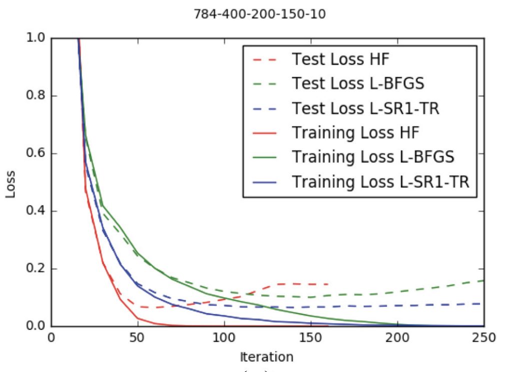
(d)



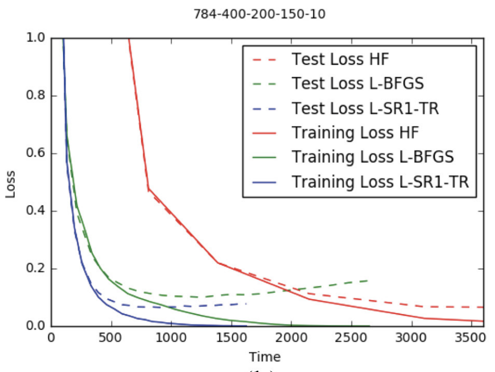
(e)



(f)



(g)



(h)

Figure 3. **Experiment IA.** Plots of the loss function versus iterations (left) and time (right) for the Hessian-Free (HF), Limited-Memory BFGS (L-BFGS), and the proposed Limited-Memory SR1 Trust-Region (L-SR1-TR) methods on the MNIST data set of handwritten digits using four different sets of hidden layers {{350,250,150}, {400,250,150,150,100,30}, {500,350}, {400,200,150}} with input layer of size 784 and output layer of size 10.

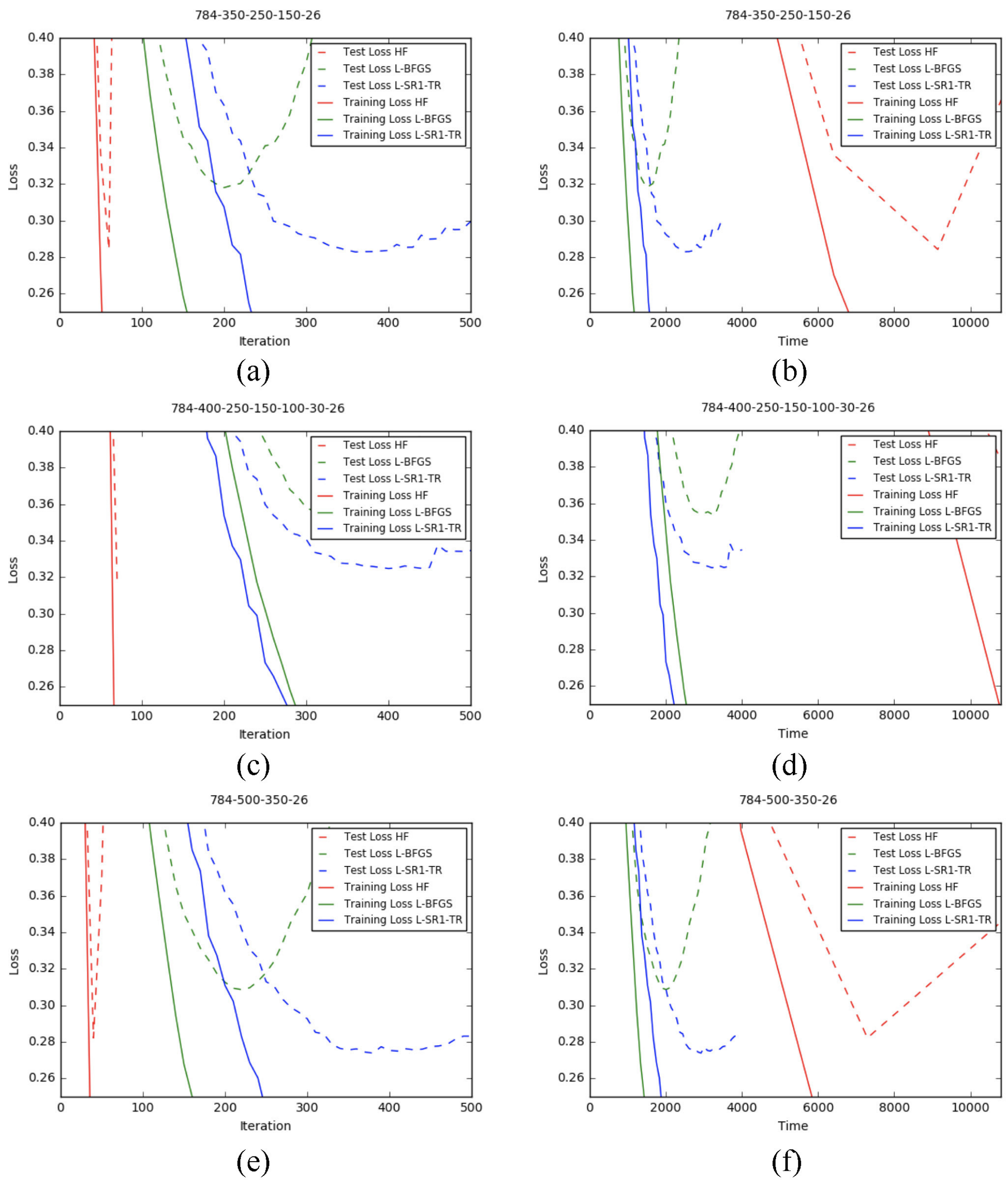
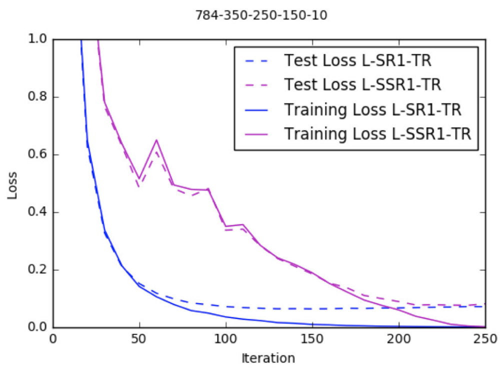
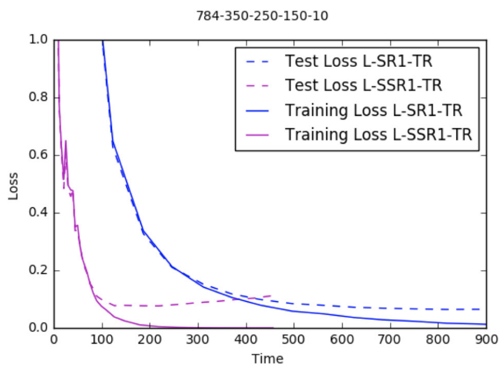


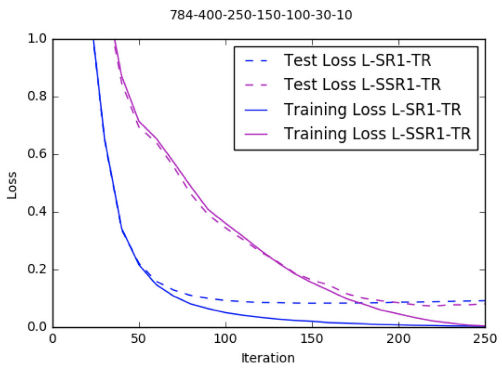
Figure 4. **Experiment IB.** Plots of the loss function versus iterations (left) and time (right) for the Hessian-Free (HF), Limited-Memory BFGS (L-BFGS), and the proposed Limited-Memory SR1 Trust-Region (L-SR1-TR) methods on the EMNIST data set of handwritten letters using three different sets of hidden layers {{350,250,150}, {400,250,150,100,30}, {500,350}} with input layer of size 784 and output layer of size 26.



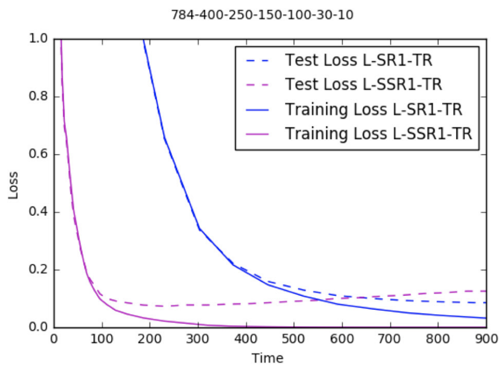
(a)



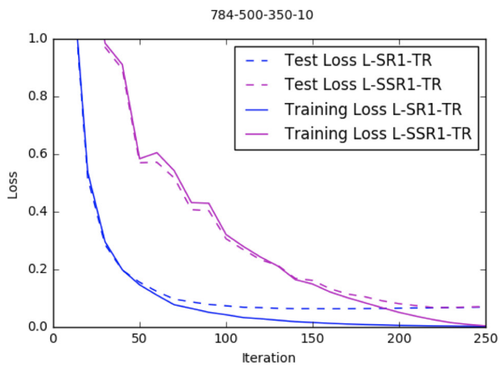
(b)



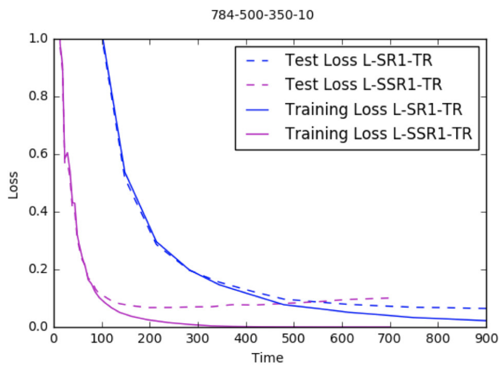
(c)



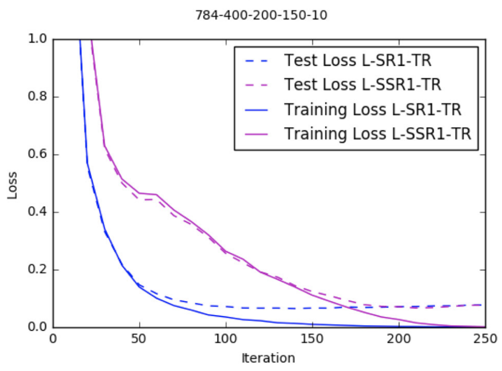
(d)



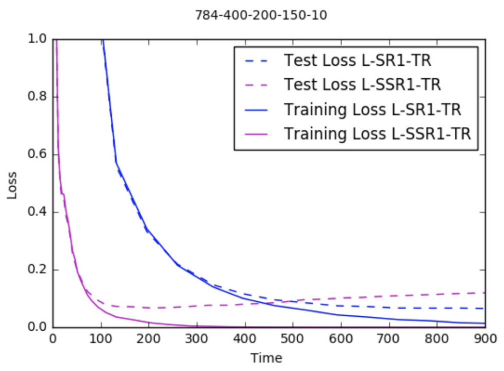
(e)



(f)

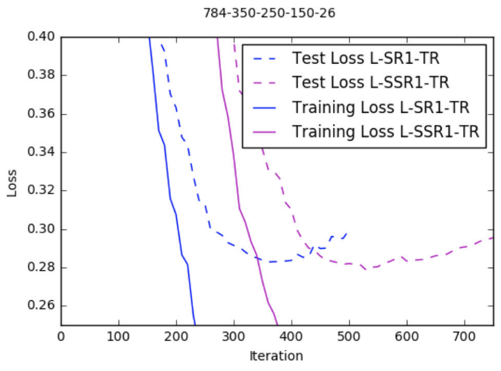


(g)

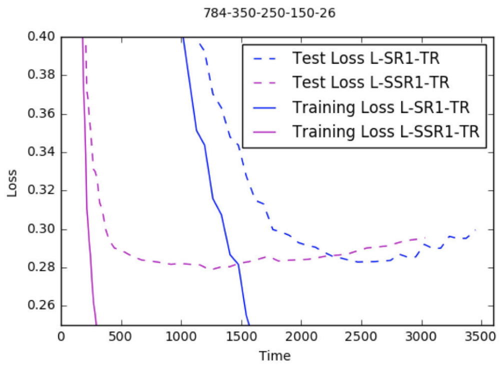


(h)

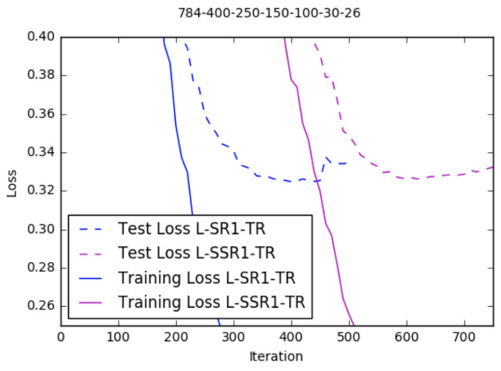
Figure 5. **Experiment IIA.** Plots of the loss function versus iterations (left) and time (right) for the proposed Limited-Memory SR1 Trust-Region (L-SR1-TR) and Limited-Memory Stochastic SR1 Trust-Region (L-SSR1-TR) methods on the MNIST data set of handwritten digits using four different sets of hidden layers {{350,250,150}, {400,250,150,150,100,30}, {500,350}, {400,200,150}} with input layer of size 784 and output layer of size 10.



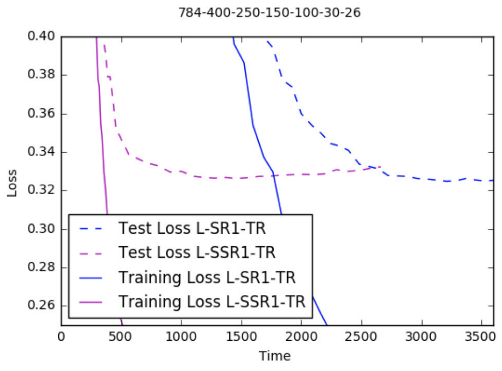
(a)



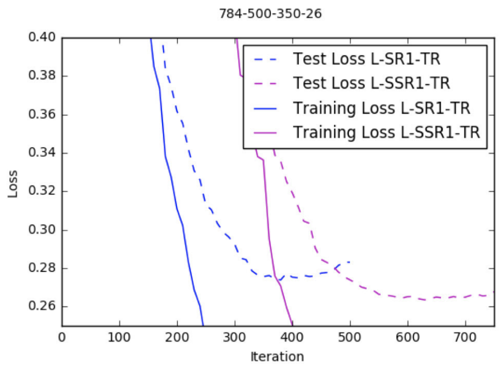
(b)



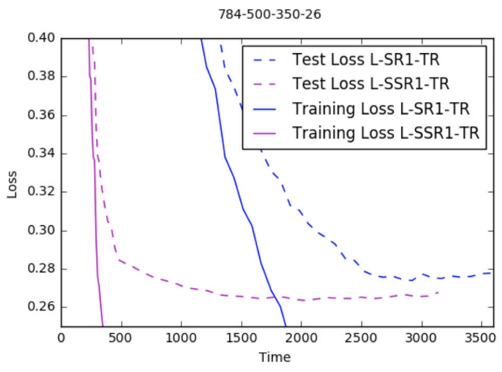
(c)



(d)



(e)



(f)

Figure 6. **Experiment IIB.** Plots of the loss function versus iterations (left) and time (right) for the proposed Limited-Memory SR1 Trust-Region (L-SR1-TR) and Limited-Memory Stochastic SR1 Trust-Region (L-SSR1-TR) methods on the EMNIST data set of handwritten letters using four different sets of hidden layers {{350,250,150}, {400,250,150,150,100,30}, {500,350}} with input layer of size 784 and output layer of size 26.