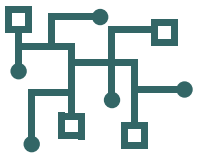

ISE

Industrial and
Systems Engineering

Quasi-Newton Methods for Deep Learning: Forget the Past, Just Sample

Albert S. Berahas[1], Majid Jahani[1], and Martin Takáč[1]

[1]Lehigh University

ISE Technical Report 19T-006

LEHIGH
UNIVERSITY

Quasi-Newton Methods for Deep Learning: Forget the Past, Just Sample

Albert S. Berahas
Lehigh Univeristy
Bethlehem, PA
albertberahas@gmail.com

Majid Jahani
Lehigh Univeristy
Bethlehem, PA
maj316@lehigh.edu

Martin Takáč
Lehigh Univeristy
Bethlehem, PA
Takac.MT@gmail.com

Abstract

We present two sampled quasi-Newton methods: sampled LBFGS and sampled LSR1. Contrary to the classical variants of these methods that sequentially build (inverse) Hessian approximations as the optimization progresses, our proposed methods sample points randomly around the current iterate to produce these approximations. As a result, the approximations constructed make use of more reliable (recent and local) information, and do not depend on past information that could be significantly stale. Our proposed algorithms are efficient in terms of accessed data points (epochs) and have enough concurrency to take advantage of distributed computing environments. We provide convergence guarantees for our proposed methods. Numerical tests on a toy classification problem and on popular benchmarking neural network training tasks reveal that the methods outperform their classical variants and are competitive with first-order methods such as ADAM.

1 Introduction

In supervised machine learning, one seeks to minimize the empirical risk,

$$\min_{w\in\mathbb{R}^d} F(w) := \frac{1}{n}\sum_{i=1}^{n} f(w; x^i, y^i) = \frac{1}{n}\sum_{i=1}^{n} f_i(w) \tag{1.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the composition of a prediction function (parametrized by w) and a loss function, and (x^i, y^i), for $i = 1, \ldots, n$, denote the training examples (samples). Difficulties arise in minimizing the function F for three main reasons: (1) the number of samples n is large; (2) the number of variables d is large; and, (3) the objective function is nonconvex.

In the last decades, much effort has been devoted to the development of stochastic first-order methods that have a low per-iteration cost, enjoy optimal complexity, are easy to implement, and that have proven to be effective for many machine learning applications. At present, the preferred method for large-scale applications is the stochastic gradient (SG) method [5, 48], and its variance-reduced [18, 26, 43, 50] and adaptive variants [19, 30]. However, these methods have several issues: (1) they are highly sensitive to the choice of hyper-parameters (e.g., steplength and batch size) and tuning can be cumbersome; (2) they suffer from ill-conditioning; and, (3) they often offer limited opportunities for parallelism; see [1, 6, 32, 49, 54].

In order to alleviate these issues, stochastic Newton [4, 9, 39, 49, 55] and stochastic quasi-Newton [2, 10, 15, 23, 27, 42, 51] methods have been proposed. These methods attempt to combine the speed of Newton's method and the scalability of first-order methods by incorporating curvature information in a judicious manner, and have proven to work well for several machine learning tasks [1, 54].

With the advances in distributed and GPU computing, it is now possible to go beyond stochastic Newton and quasi-Newton methods and use large batches to compute function, gradient and Hessian

Preprint. Under review.

vector products in order to train machine learning models. In the large batch regime, one can take advantage of parallel and distributed computing and fully utilize the capabilities of GPUs. However, researchers have observed that *well-tuned* first-order methods (e.g., ADAM) are more effective than full batch methods (e.g., LBFGS) for large-scale applications [24, 28].

Nevertheless, in this paper we focus on (full) batch methods that incorporate local second-order (curvature) information of the objective function. These methods mitigate the effects of ill-conditioning, avoid or diminish the need for hyper-parameter tuning, have enough concurrency to take advantage of parallel computing, and, due to requiring fewer iterations enjoy low communication costs. Specifically, we focus on quasi-Newton methods [45]; methods that construct curvature information using first-order (gradient) information. We propose two variants of classical quasi-Newton methods that sample a small number of random points at every iteration to build (inverse) Hessian approximations.

We are motivated by the results presented in Figure 1 that illustrate the performance (for 10 different starting points) of several stochastic and deterministic, first- and second-order methods on a toy neural network classification task, given budget; see Section 6 for details. As is clear from the results, first-order methods converge very slowly, and sometimes even fail to achieve 100% accuracy. Similarly, classical quasi-Newton methods are also slow or stagnate. On the other hand, methods that use the true Hessian are able to converge in very few iterations from all starting points. This seems to suggest that for some neural network training tasks second-order information is important, and that the curvature information captured by classical quasi-Newton methods may not be adequate or useful.

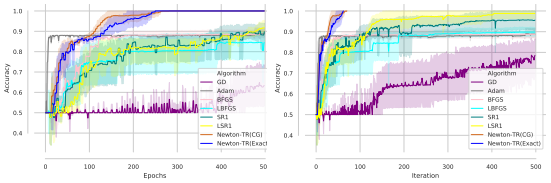

Figure 1: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, Newton-TR(CG, Exact) on a toy classification problem.

The key idea of our proposed methods is to leverage the fact that quasi-Newton methods can incorporate second-order information using only gradient information at a reasonable cost, but at the same time to enhance the (inverse) Hessian approximations by using more reliable (recent and local) information. The fundamental component of our methods, and what differentiates them from the classical variants, is the manner in which the curvature pairs are constructed. To this end, we propose to *forget* past curvature information and *sample* new curvature pairs at every iteration.

Contributions Our contributions can be summarized as follows:

- We propose two novel quasi-Newton methods that use sampling to construct Hessian approximations. We analyze the convergence properties of both methods, and show that their theoretical guarantees match those of their classical limited memory counterparts.
- We discuss the implementation costs of the sampled quasi-Newton methods and compare them to the classical variants, and illustrate the scaling properties of the methods compared to the SG method on distributed computing platforms on real large-scale network architectures.
- We illustrate the practical performance of the methods on a toy classification problem and on standard benchmarking neural network training tasks, and show their advantages over the classical variants. We posit that this is the case since the (inverse) Hessian approximations constructed by our proposed methods capture better (more informative) curvature information. Moreover, the proposed methods are easily parallelizable and efficient in terms of iteration, epochs and communication.

The paper is organized as follows. We conclude this section with a literature review of quasi-Newton methods. We describe the classical (L)BFGS and (L)SR1 methods in Section 2, and in Section 3 we detail our proposed sampled variants. In Section 4, we discuss the computational cost of the proposed methods and show their scaling properties. We show the theoretical properties of our proposed methods in Section 5. Numerical results on neural network training tasks are reported in Section 6. Finally, in Section 7 we provide some final remarks and discuss several avenues for future work.

Brief Literature Review Quasi-Newton methods, such as BFGS [7, 20, 22, 52] and SR1 [11, 13, 29] and their limited-memory variants LBFGS [36, 44] and LSR1 [8, 38], respectively, have been studied extensively in the deterministic nonlinear optimization literature. These methods incorporate curvature (second-order) information using only gradient (first-order) information, have good theoretical guarantees, and have proven to be effective in practice.

In the context of deep neural networks, both full batch and stochastic quasi-Newton methods seem to perform worse than (stochastic) first-order methods. Nevertheless, several stochastic quasi-Newton methods have been proposed; see e.g., [3, 10, 51]. What distinguishes these methods from one another is the way in which curvature pairs are constructed. Our methods borrow some of the ideas proposed in [10, 23, 37]. Specifically, we use Hessian vector products in lieu of gradient displacements.

Possibly the closest works to ours are Block BFGS [21] and its stochastic variant [23]. These methods construct multiple curvature pairs to update the quasi-Newton matrices. However, there are several key features that are different from our approach; in these works (1) the Hessian approximation is not updated at every iteration, and (2) they enforce that multiple secant equations hold simultaneously.

2 Quasi-Newton Methods

In this section, we review two classical quasi-Newton methods and their limited memory variants. This will set the stage for our proposed sampled quasi-Newton methods.

BFGS and LBFGS Let us begin by considering the BFGS method and then consider its limited memory version. At the kth iteration, the BFGS method computes a new iterate by the formula

$$w_{k+1} = w_k - \alpha_k H_k \nabla F(w_k), \tag{2.1}$$

where α_k is the step length, $\nabla F(w_k)$ is the gradient of (1.1) and H_k is the inverse BFGS Hessian approximation that is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T,$$

where $\rho_k = 1/y_k^T s_k$, $V_k = I - \rho_k y_k s_k^T$ and the curvature pairs (s_k, y_k) are defined as

$$s_k = w_k - w_{k-1}, \quad y_k = \nabla F(w_k) - \nabla F(w_{k-1}). \tag{2.2}$$

As is clear, the curvature pairs (2.2) are constructed sequentially (at every iteration), and as such the inverse Hessian approximation at the kth iteration H_k depends on past iterate (and gradient) information. The inverse BFGS Hessian approximations are constructed to satisfy two conditions: the secant ($H_{k+1} y_k = s_k$) and curvature ($s_k^T y_k > 0$) conditions, as well as symmetry. Consequently, as long as the initial inverse Hessian approximation is positive definite, then all subsequent inverse BFGS Hessian approximations are also positive definite. Note, the new inverse Hessian approximation H_{k+1} differs from the old approximation H_k by a rank-2 matrix.

In the limited memory version, the matrix H_k is defined at each iteration as the result of applying m BFGS updates to a multiple of the identity matrix using the set of m most recent curvature pairs $\{s_i, y_i\}$ kept in storage. As a result, one need not construct and store the dense inverse Hessian approximation, rather one can store two $m \times d$ matrices and compute the matrix-vector product in (2.1) via the two-loop recursion [45].

SR1 and LSR1 Contrary to the BFGS updating formula, and as suggested by the name, the symmetric-rank-1 (SR1) updating formula allows one to satisfy the secant equation and maintain symmetry with a simpler rank-1 update. However, unlike BFGS, the SR1 update does not guarantee that the updated matrix maintains positive definiteness. As such, the SR1 method is usually implemented with a trust region; we introduce it in this way below.

At the kth iteration, the SR1 method computes a new iterate by the formula

$$w_{k+1} = w_k + p_k, \tag{2.3}$$

where p_k is the minimizer of the following subproblem

$$\min_{\|p\| \leq \Delta_k} m_k(p) = F(w_k) + \nabla F(w_k)^T p + \tfrac{1}{2} p^T B_k p, \tag{2.4}$$

Δ_k is the trust region radius and B_k is the SR1 Hessian approximation computed as

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \tag{2.5}$$

Similar to LBFGS, in the limited memory version of SR1 the matrix B_k is defined as the result of applying m SR1 updates to a multiple of the identity matrix, using a set of m correction pairs $\{s_i, y_i\}$ kept in storage.

3 Sampled Quasi-Newton Methods

In this section, we describe our two proposed sampled quasi-Newton methods; S-LBFGS and S-LSR1. The main idea of these methods, and what differentiates them from the classical variants, is the way in which curvature pairs are constructed. At every iteration, a small number (m) of points are sampled around the current iterate and used to construct a new set of curvature pairs. In other words, contrary to the sequential nature of classical quasi-Newton methods, our proposed methods *forget* all past curvature pairs and construct new curvature pairs from scratch via *sampling*.

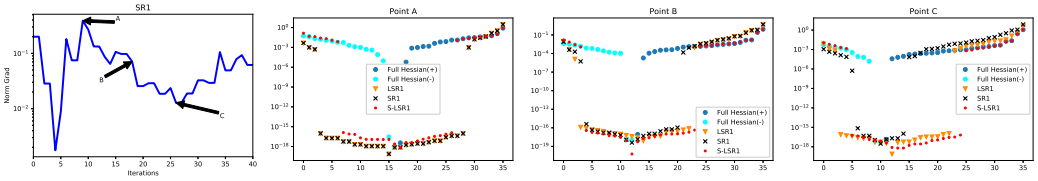

Figure 2: Comparison of the eigenvalues of (L)SR1 and S-LSR1 (@ A, B, C) for a toy classification problem.

Our motivation stems from the following observation: by constructing Hessian approximations via sampling, one is able to better capture curvature information of the objective function. In Figure 2, we show the spectrum of the true Hessian, and compare it to the spectra of different SR1 Hessian approximations at several points. As is clear from the results, the eigenvalues of the S-LSR1 Hessian approximations better match the eigenvalues of the true Hessian compared to the eigenvalues of the SR1 and LSR1 Hessian approximations. This is not surprising since S-LSR1 uses newly sampled local information, and unlike the classical variants does not rely on past information that could be significantly stale. Similar results were obtained for other problems; see Appendix B.2 for details.

This, of course, does not come for free. The classical variants construct curvature pairs as the optimization progresses at no additional cost, whereas the sampled quasi-Newton methods require the construction of m new curvature pairs at every iteration. We discuss implementation issues and the computational cost of the sampled quasi-Newton methods in Sections 3.1 and 4.

3.1 Sampling Curvature Pairs

As mentioned above, the key component of our proposed algorithms is the way in which curvature pairs are constructed. A pseudo-code of our proposed sampling strategy is given in Algorithm 1. Let $S, Y \in \mathbb{R}^{d\times m}$ denote the matrices of all curvature pairs constructed during the kth iteration.

Algorithm 1 Compute new (S, Y) curvature pairs

Input: w (iterate), m (memory), r (sampling radius), $S = [\,]$, $Y = [\,]$ (curvature pair containers).

1: Compute $\nabla F(w)$
2: **for** $i = 1, 2, ..., m$ **do**
3: Sample a random direction σ_i
4: Construct $\bar{w} = w + r\sigma_i$
5: Set $s = w - \bar{w}$ and

$$y = \begin{cases} \nabla F(w) - \nabla F(\bar{w}), & \text{Option I} \\ \nabla^2 F(w)s, & \text{Option II} \end{cases}$$

6: Set $S = [S\ s]$ and $Y = [Y\ y]$
7: **end for**

Output: $S, Y, \nabla F(w)$

At every iteration, given the current iterate and gradient, m curvature pairs are constructed. The subroutine first samples points around the current iterate along random directions σ_i and sets the iterate displacement curvature pair (s), and then creates the gradient difference curvature pair (y) via gradient differences (Option I) or Hessian vector products (Option II).

Our theory holds for both options; however, in our numerical experiments we present results with Option II only for the following reasons. Option I requires m gradient evaluations (m epochs), and thus requires accessing the data m times. On the other hand, Option II only requires a single Hessian matrix product which can be computed very efficiently on a GPU, as the y curvature pairs can be constructed simultaneously, i.e., $Y = \nabla^2 F(w)S$, and thus only requires accessing the data once. Moreover, Option I requires choosing the sampling radius r, whereas Option II does not since it is scale invariant.

3.2 Sampled LBFGS (S-LBFGS)

At the kth iteration, the S-LBFGS method (Algorithm 2) computes a new iterate via (2.1), where the inverse Hessian approximation is constructed using the curvature pairs sampled by Algorithm 1.

Algorithm 2 Sampled LBFGS (S-LBFGS)

Input: w_0 (initial iterate), m (memory), r (sampling radius).

1: **for** $k = 0, 1, 2, \ldots$ **do**
2: Compute new (S_k, Y_k) pairs via Algorithm 1
3: Compute the search direction $p_k = -H_k \nabla F(w_k)$
4: Choose the steplength $\alpha_k > 0$
5: Set $w_{k+1} = w_k + \alpha_k p_k$
6: **end for**

Algorithm 2 is almost identical to the classical (L)BFGS algorithm [45]; however, it has two key differentiating features: (1) the way in which curvature pairs are created; and, (2) the location in the algorithm where the curvature pairs are constructed. First, using a similar argument as that for the S-LSR1 method, the inverse Hessian approximations constructed by this method better capture local curvature information of the objective function. Moreover, notice that the first set of curvature pairs is constructed before a single step is taken by the method (Line 2). This allows the method to take quasi-Newton-type (well-scaled) steps from the first iteration, which is not the case for classical BFGS methods that usually take a gradient-type step in the first iteration and in which imposing the correct scale is always an issue. This, possibly, is a more important feature of the method, as the first step taken by quasi-Newton methods can be of paramount importance.

3.3 Sampled LSR1

Algorithm 3 Sampled LSR1 (S-LSR1)

Input: w_0 (initial iterate), Δ_0 (initial trust region radius), m (memory), r (sampling radius).

1: **for** $k = 0, 1, 2, \ldots$ **do**
2: Compute new (S_k, Y_k) pairs via Algorithm 1
3: Compute p_k by solving the subproblem (2.4)
4: Compute $\rho_k = \frac{F(w_k) - F(w_k + p_k)}{m_k(0) - m_k(p_k)}$
5: **if** $\rho_k \geq \eta_1$ **then**
6: Set $w_{k+1} = w_k + p_k$
7: **else**
8: Set $w_{k+1} = w_k$
9: **end if**
10: $\Delta_{k+1} = \texttt{adjustTR}(\Delta_k, \rho_k)$ [see Appendix B.3]
11: **end for**

At the kth iteration, the S-LSR1 method computes a new iterate via (2.3), where the Hessian approximation in (2.4) is constructed using the curvature pairs sampled by Algorithm 1. The S-LBFGS method is outlined in Algorithm 3.

The S-LSR1 method has the same key features as S-LBFGS that differentiates it from the classical SR1 methods. The subroutine `adjustTR` (Step 12, Algorithm 3) adjusts the trust-region based on the progress made by the method. For brevity we omit the details of this subroutine, and refer the reader to Appendix B.3 for the details.

4 Distributed Computing and Computational Cost

In this section, we show the scalability and computation cost of the sampled quasi-Newton methods.

Distributed Computing In Figure 3 (left), we show how the batch size affects the number of data points processed per second to compute the function, gradient and Hessian vector products on a NVIDIA Tesla P100 GPU for various deep neural networks; see Appendix C.2. Using small batch sizes one is not able to fully utilize the power of GPUs; however, using larger batches in conjunction with SG-type algorithms does not necessarily reduce training time [17, 53]. Moreover, we observe that for these networks the cost of computing function values, gradients and Hessian vector products is comparable[1]. In Figure 3 (bar graphs), we compare the time to perform 1 epoch of SG (assuming we have 1M images) with the time to perform 1 iteration of S-LSR1. For SG, we show results for different batch sizes on each GPU[2]: (1) batch size 16 (SGD 16); and, (2) batch size 32, 64 and 128 for vgg a, LeNet and alexnet v2, respectively, (SGD Default). The reason there is no significant benefit when using more GPUs for SG is that the cost is dominated by the communication. This is not the case for S-LSR1; significant performance gains can be achieved by scaling up the number of MPI processes since much less communication is involved. See Section C.1 for more details.

Cost, Storage and Parallelization The cost per iteration of quasi-Newton methods can be decomposed as follows: (1) cost of computing the gradient, and (2) cost of forming the search direction and taking the step. The gradient computation is common amongst the quasi-Newton methods, whereas

[1] We assume that the cost of computing function values, gradients and Hessian vector products is $\mathcal{O}(nd)$.

[2] Each GPU has 1 MPI process that is used for communicating updates. Note, we are running 4 MPI processes for each physical node, i.e., each node has 4 P100 GPUs

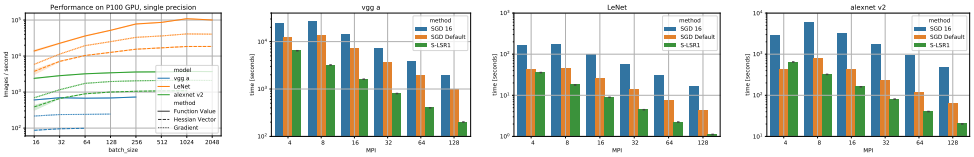

Figure 3: Performance (Images/second) as a function of batch size for different DNN models and operations on a single NVIDIA Tesla P100 GPU (left). Time (sec) to complete 1 epoch of SG and to perform 1 iteration of S-LSR1 on a dataset with 1M images using varying number of MPI processes (bar charts).

the search directions are computed differently. Specifically, for BFGS methods we employ a line search and for SR1 methods we use a trust region and solve the subproblem (2.4) using CG; see [45]. The sampled quasi-Newton methods do not have a significantly higher cost per iteration than the classical limited memory variants. In the regime where $m \ll n, d$, the computational cost is $\mathcal{O}(nd)$. Moreover, several computations that are extra in our proposed methods (e.g., the construction of the gradient displacement pairs y) are easily parallelizeable. The computational cost and storage for the different quasi-Newton methods are summarized in Table 4; see Section C.3 for more details.

5 Convergence Analysis

In this section, we present convergence analyses for the sampled quasi-Newton methods. For brevity, we omit the proofs from the paper; see Appendix A for the proofs.

5.1 Sampled LBFGS

Strongly Convex Functions We make the following standard assumptions.

Assumption 5.1. *F is twice continuously differentiable.*

Assumption 5.2. *There exist constants $0 < \mu \leq L$ such that $\mu I \preceq \nabla^2 F(w) \preceq LI$, for all $w \in \mathbb{R}^d$.*

First, we show that the inverse Hessian approximations generated by the S-LBFGS method have eigenvalues that are uniformly bounded above and away from zero. The proof technique is an adaptation of that in [2, 10]; however, modifications are necessary since the inverse Hessian approximations are constructed using information only from the current iterate, and not constructed sequentially.

Lemma 5.3. *If Assumptions 5.1 and 5.2 hold, there exist constants $0 < \mu_1 \leq \mu_2$ such that the inverse Hessian approximations $\{H_k\}$ generated by Algorithm 2 satisfy, $\mu_1 I \preceq H_k \preceq \mu_2 I$.*

We show that the sampled LBFGS method with a constant step length converges to the optimal solution at a linear rate (Theorem 5.4). This result is similar in nature to the result for LBFGS [36].

Theorem 5.4. *Suppose that Assumptions 5.1 and 5.2 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 2, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for all $k \geq 0$, $F(w_k) - F^\star \leq \left(1 - \alpha\mu\mu_1\right)^k [F(w_0) - F^\star]$.*

Nonconvex Functions For nonconvex functions, it is known that the (L)BFGS method can fail [16, 40]. To establish convergence in the nonconvex setting several techniques have been proposed [34, 35, 47]. Here we employ a *cautious strategy* that is well suited to our particular algorithm; we update the inverse Hessian approximation using only the set of curvature pairs that satisfy

$$s^T y \geq \epsilon \|s\|^2, \tag{5.1}$$

where $\epsilon > 0$ is a predetermined constant. Using said mechanism we prove that the eigenvalues of the inverse Hessian approximations generated by the S-LBFGS method are bounded above and away from zero. For this analysis, we make the following assumptions in addition to Assumption 5.1.

Assumption 5.5. *The function $F(w)$ is bounded below by a scalar $\widehat{F}$.*

Assumption 5.6. *The gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Lemma 5.7. *Suppose that Assumptions 5.1 and 5.6 hold. Let $\{H_k\}$ be the inverse Hessian approximations generated by Algorithm 2, with the modification that the inverse approximation update is*

performed using only curvature pairs that satisfy (5.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (5.1). Then, there exist constants $0 < \mu_1 \leq \mu_2$ such that, $\mu_1 I \preceq H_k \preceq \mu_2 I$.

We show that S-LBFGS with *cautious updating* converges to a stationary point (Theorem 5.8).

Theorem 5.8. *Suppose that Assumptions 5.1, 5.5 and 5.6 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 2, with the modification that the inverse Hessian approximation update is performed using only curvature pairs that satisfy (5.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (5.1), where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for any $\tau > 1$, $\frac{1}{\tau}\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0)-\widehat{F}]}{\alpha\mu_1\tau} \xrightarrow{\tau\to\infty} 0$.*

5.2 Sampled LSR1

In order to establish convergence results one needs to ensure that the SR1 Hessian update equation (2.5) is well defined. To this end, we employ a *cautious updating mechanism*; we update the Hessian approximation using only the set of curvature pairs that satisfy

$$|s^T(y - Bs)| \geq \epsilon \|s\| \|y - Bs\|, \tag{5.2}$$

where $\epsilon > 0$ is a predetermined constant. It is not trivial to test this condition in practice without explicitly constructing $d \times d$ matrices. We discuss this in detail in Appendix B.4.

For the analysis in this section, we make the following assumption that implies that at every iteration the trust-region subproblem is solved sufficiently accurately.

Assumption 5.9. *For all k, $m_k(0) - m_k(p_k) \geq \xi \|\nabla F(w_k)\| \min\left[\frac{\|\nabla F(w_k)\|}{\beta_k}, \Delta_k\right]$, where $\xi \in (0,1)$ and $\beta_k = 1 + \|B_k\|$.*

We prove that the Hessian approximations B_k generated by the S-LSR1 method are uniformly bounded from above. The proof technique is an adaptation of that in [38]; however, modifications are necessary since the Hessian approximations are constructed using information only from the current iterate, and not constructed sequentially.

Lemma 5.10. *Suppose that Assumptions 5.1, 5.6 and 5.9 hold. Let $\{B_k\}$ be the Hessian approximations generated by Algorithm 3, with the modification that the approximation update is performed using only curvature pairs that satisfy (5.2), for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (5.2). Then, there exists a constant $\nu_2 > 0$ such that $\|B_k\| \leq \nu_2$.*

We show that the S-LSR1 with *cautious updating* converges to a stationary point (Theorem 5.11). This result is similar in nature to that in [38]. In order to prove the following result, we make use of well-known results for Trust-Region methods; see [14].

Theorem 5.11. *Suppose that Assumptions 5.1, 5.5, 5.6 and 5.9 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 3, with the modification that the Hessian approximation update is performed using only curvature pairs that satisfy (5.2), for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (5.2). Then, $\lim_{k\to\infty} \|\nabla F(w_k)\| = 0$.*

6 Numerical Experiments

In this section, we present numerical experiments on a toy classification problem and on popular benchmarking neural network training tasks[3]. See Appendix B.5 for implementation details.

6.1 A Toy Classification Problem

Consider the simple classification problem, illustrated in Figure 4, consisting of two classes each with 50 data points. We trained three fully connected neural networks–`small`, `medium` and `large`–with sigmoid activation functions and 4 hidden layers; see Appendix B.6, Table 1 for details. For this problem, we ran

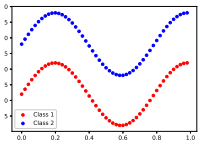

Figure 4: Toy Problem

[3]All codes to reproduce the results presented in this section are available at: http://github.com/ANONYMOUS/LINK. The code will be released upon acceptance of the paper.

each method 100 times starting from different initial points and show the results for different budget levels in Figure 5. As is clear from the figures, the proposed methods outperform their classical variants as well as the first-order methods. See Appendix B.6 for more results.

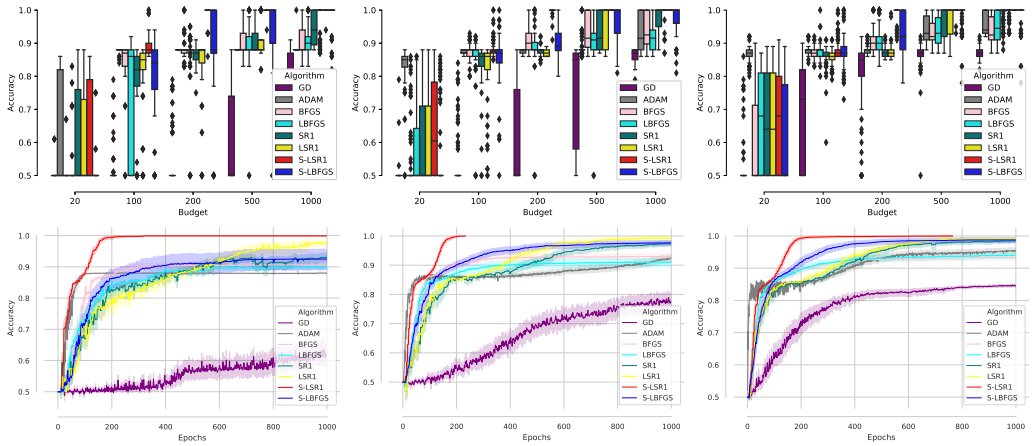

Figure 5: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problems. Networks: `small` (left); `medium` (right); `large` (right).

6.2 MNIST and CIFAR10

We illustrate the performance of the sampled quasi-Newton methods on standard benchmarking neural network training tasks: `MNIST` [33] and `CIFAR10` [31]. The details of the problems are given in Appendix B.7, Table 2. For these problems we used sigmoid activation functions and softmax cross-entropy loss. The results of these experiments are given in Figure 6. Overall, the sampled quasi-Newton methods outperform their classical variants. For the `MNIST` problem, the S-LSR1 method is able to achieve comparable accuracy to that of *well-tuned* ADAM, after a lot more epochs. That being said, in a distributed setting, the time to perform one iteration (one epoch) of S-LSR1 is significantly smaller than the time to perform one epoch of ADAM, and as such in terms of Wall Clock Time, the proposed method could be more efficient. Moreover, ADAM requires meticulous tuning (see Appendix B.7) whereas S-LSR1 is parameter-free. We posit that similar observations could be made for `CIFAR10` if the experiments were run longer.

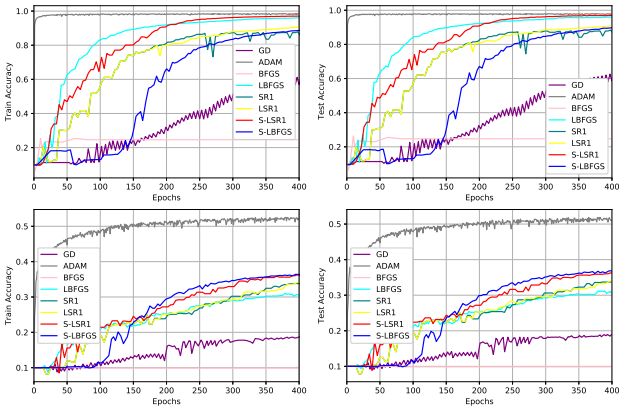

Figure 6: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on `MNIST` (top) and `CIFAR10` (bottom).

7 Final Remarks and Future Work

This paper describes two novel quasi-Newton methods; S-LBFGS and S-LSR1. Contrary to classical quasi-Newton methods, these methods *forget* past curvature information and *sample* new curvature information at every iteration. Numerical results show that the methods are efficient in practice, and the convergence guarantees of the methods match those of the classical variants.

Our algorithms can be extended to the stochastic setting where gradients and/or Hessians are computed inexactly. Moreover, the algorithms could be made adaptive following the ideas from [25, 41]. Furthermore, stronger theoretical (e.g., superlinear convergence) results could be proven for some variants of the sampled quasi-Newton methods. Finally, a large-scale numerical investigation would test the limits of the methods.

Acknowledgements

This work was partially supported by the U.S. National Science Foundation, under award numbers NSF:CCF:1618717, NSF:CMMI:1663256 and NSF:CCF:1740796, DARPA Lagrange award HR-001117S0039, and XSEDE Startup grant IRI180020.

References

[1] Albert S Berahas, Raghu Bollapragada, and Jorge Nocedal. An investigation of newton-sketch and subsampled newton methods. *arXiv preprint arXiv:1705.06211*, 2017.

[2] Albert S Berahas, Jorge Nocedal, and Martin Takác. 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. *arXiv preprint arXiv:1707.08552*, 2017.

[4] Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis*, 2016.

[5] Léon Bottou and Yann L Cun. Large scale online learning. In *Advances in neural information processing systems*, pages 217–224, 2004.

[6] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[7] Charles G Broyden. Quasi-newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[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] Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

[10] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[11] Richard H Byrd, Humaid Fayez Khalfan, and Robert B Schnabel. Analysis of a symmetric rank-one trust region method. *SIAM Journal on Optimization*, 6(4):1025–1039, 1996.

[12] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Math. Program.*, 63:129–156, 1994.

[13] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. Convergence of quasi-newton matrices generated by the symmetric rank one update. *Mathematical programming*, 50(1-3):177–195, 1991.

[14] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*, volume 1. Siam, 2000.

[15] Frank Curtis. A self-correcting variable-metric algorithm for stochastic optimization. In *International Conference on Machine Learning*, pages 632–641, 2016.

[16] Yu-Hong Dai. Convergence properties of the bfgs algoritm. *SIAM Journal on Optimization*, 13(3):693–701, 2002.

[17] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*, 2016.

[18] 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.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[20] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[21] Wenbo Gao and Donald Goldfarb. Block bfgs methods. *SIAM Journal on Optimization*, 28(2):1205–1231, 2018.

[22] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[23] Robert Gower, Donald Goldfarb, and Peter Richtárik. Stochastic block bfgs: Squeezing more curvature out of data. In *International Conference on Machine Learning*, pages 1869–1878, 2016.

[24] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: stability of stochastic gradient descent. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 1225–1234. JMLR. org, 2016.

[25] Majid Jahani, Xi He, Chenxin Ma, Aryan Mokhtari, Dheevatsa Mudigere, Alejandro Ribeiro, and Martin Takáč. Efficient distributed hessian free algorithm for large-scale empirical risk minimization via accumulating sample strategy. *arXiv preprint arXiv:1810.11507*, 2018.

[26] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[27] Nitish Shirish Keskar and Albert S Berahas. adaqn: An adaptive quasi-newton algorithm for training rnns. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 1–16. Springer, 2016.

[28] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[29] H Fayez Khalfan, Richard H Byrd, and Robert B Schnabel. A theoretical and experimental study of the symmetric rank-one update. *SIAM Journal on Optimization*, 3(1):1–24, 1993.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[32] Sudhir B Kylasa, Farbod Roosta-Khorasani, Michael W Mahoney, and Ananth Grama. Gpu accelerated sub-sampled newtons method. *arXiv preprint arXiv:1802.09113*, 2018.

[33] 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.

[34] Dong-Hui Li and Masao Fukushima. A modified bfgs method and its global convergence in nonconvex minimization. *Journal of Computational and Applied Mathematics*, 129(1-2):15–35, 2001.

[35] Dong-Hui Li and Masao Fukushima. On the global convergence of the bfgs method for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, 11(4):1054–1064, 2001.

[36] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[37] Jie Liu, Yu Rong, Martin Takac, and Junzhou Huang. On the acceleration of l-bfgs with second-order information and stochastic batches. *arXiv preprint:1807.05328*, 2018.

[38] Xuehua Lu. *A study of the limited memory SR1 method in practice*. University of Colorado at Boulder, 1996.

[39] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[40] Walter F Mascarenhas. The bfgs method with exact line searches fails for non-convex objective functions. *Mathematical Programming*, 99(1):49–61, 2004.

[41] Aryan Mokhtari, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, and Alejandro Ribeiro. Adaptive newton method for empirical risk minimization to statistical accuracy. In *Advances in Neural Information Processing Systems 29*, pages 4062–4070. 2016.

[42] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[43] Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *International Conference on Machine Learning*, pages 2613–2621, 2017.

[44] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[45] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, second edition, 2006.

[46] Michael JD Powell. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. *Nonlinear programming*, 9(1):53–72, 1976.

[47] Michael JD Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14(1):224–248, 1978.

[48] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[49] Farbod Roosta-Khorasani and Michael W. Mahoney. Sub-sampled newton methods. *Mathematical Programming*, 2018.

[50] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[51] 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, 2007.

[52] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[53] Martin Takác, Avleen Singh Bijral, Peter Richtárik, and Nati Srebro. Mini-batch primal and dual methods for svms. In *ICML (3)*, pages 1022–1030, 2013.

[54] Peng Xu, Farbod Roosta-Khorasan, and Michael W Mahoney. Second-order optimization for non-convex machine learning: An empirical study. *arXiv preprint arXiv:1708.07827*, 2017.

[55] Peng Xu, Farbod Roosta-Khorasani, and Michael W Mahoney. Newton-type methods for non-convex optimization under inexact hessian information. *arXiv preprint arXiv:1708.07164*, 2017.

A Theoretical Results and Proofs

We first restate the Assumptions that we use in the Convergence Analysis section (Section 5). We them prove all the results that appear in the main paper (Lemmas 5.3, 5.7 & 5.10; Theorems 5.4, 5.8 & 5.11).

A.1 Assumptions

Assumption 5.1. *F is twice continuously differentiable.*

Assumption 5.2. *There exist positive constants μ and L, such that*

$$\mu I \preceq \nabla^2 F(w) \preceq LI,$$

for all $w \in \mathbb{R}^d$.

Assumption 5.5. *The function $F(w)$ is bounded below by a scalar $\widehat{F}$.*

Assumption 5.6. *The gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Assumption 5.9. *For all k,*

$$m_k(0) - m_k(p_k) \geq \xi \|\nabla F(w_k)\| \min\left[\frac{\|\nabla F(w_k)\|}{\beta_k}, \Delta_k\right],$$

where $\xi \in (0,1)$ and $\beta_k = 1 + \|B_k\|$.

A.2 Proof of Lemma 5.3

Lemma 5.3. *If Assumptions 5.1 and 5.2 hold, there exist constants $0 < \mu_1 \leq \mu_2$ such that the inverse Hessian approximations $\{H_k\}$ generated by Algorithm 2 satisfy,*

$$\mu_1 I \preceq H_k \preceq \mu_2 I, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{A.1}$$

Proof. First, note that there is a chance that no curvature pairs are selected in Algorithm 1. In this case, the inverse Hessian approximation is $H_k = I$, and thus $\mu_1 = \mu_2 = 1$ and condition (A.1) is satisfied.

We now consider the case where at least one curvature pair is selected by Algorithm 1. Instead of analyzing the inverse Hessian approximation H_k, we study the direct Hessian approximation $B_k = H_k^{-1}$. In this case, the sampled LBFGS updating formula is given as follows. Let $\tilde{m}_k \in \{1, ..., m\}$ denote the number of curvature pairs that satisfy (5.1) at the kth iteration, where m is the memory. At the kth iteration, given a set of curvature pairs $(s_{k,j}, y_{k,j})$, for $j = 1, \ldots, \tilde{m}_k$

1. Set $B_k^{(0)} = \frac{y_{k,l}^T y_{k,l}}{s_{k,l}^T y_{k,l}} I$, where l is chosen uniformly at random from $\{1, \ldots, \tilde{m}_k\}$.
2. For $i = 1, \ldots, \tilde{m}_k$ compute

$$B_k^{(i)} = B_k^{(i-1)} - \frac{B_k^{(i-1)} s_{k,i} s_{k,i}^T B_k^{(i-1)}}{s_{k,i}^T B_k^{(i-1)} s_{k,i}} + \frac{y_{k,i} y_{k,i}^T}{y_{k,i}^T s_{k,i}}.$$

3. Set $B_{k+1} = B_k^{(\tilde{m}_k)}$.

In our algorithm (Algorithm 1), there are two options for constructing the curvature pairs $s_{k,j}$ and $y_{k,j}$. At the current iterate w_k we sample points $\bar{w}_j$ for $j = 1, \ldots, m$ and set

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla F(w_k) - \nabla F(\bar{w}_j) \qquad \text{Option I,} \tag{A.2}$$

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla^2 F(w_k) s_k \qquad \text{Option II.} \tag{A.3}$$

We now prove an upper and lower bound for $\frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}}$, for all $j = 1, \ldots \tilde{m}_k$, for both options.

Option I: A consequence of Assumption 5.2 is that the eigenvalues of the Hessian matrix are bounded above and away from zero. Utilizing this fact, the convexity of the objective function and the definitions (A.2), we have

$$y_{k,j}^T s_{k,j} \geq \frac{1}{L}\|y_{k,j}\|^2 \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \leq L. \tag{A.4}$$

On the other hand, strong convexity of the sub-sampled functions, the consequence of Assumption 5.2 and definitions (A.2), provide a lower bound,

$$y_{k,j}^T s_{k,j} \leq \frac{1}{\mu}\|y_{k,j}\|^2 \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \geq \mu. \tag{A.5}$$

Combining the upper and lower bounds (A.4) and (A.5)

$$\mu \leq \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \leq L. \tag{A.6}$$

Option II: A consequence of Assumption 5.2 is that the eigenvalues of the Hessian matrix are bounded above and away from zero. Utilizing this fact and the definitions (A.2), we have

$$\mu\|s_{k,j}\|^2 \leq y_{k,j}^T s_{k,j} = s_{k,j}^T \nabla^2 F(w_k) s_{k,j} \leq L\|s_{k,j}\|^2. \tag{A.7}$$

We have that,

$$\frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} = \frac{s_{k,j}^T \nabla^2 F(w_k)^2 s_{k,j}}{s_{k,j}^T \nabla^2 F(w_k) s_{k,j}}, \tag{A.8}$$

and since $\nabla^2 F(w_k)$ is symmetric and positive definite, it has a square root and so

$$\mu \leq \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \leq L. \tag{A.9}$$

The bounds on $\frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}}$ prove that for any l chosen uniformly at random from $\{1, \ldots, \tilde{m}_k\}$ the eigenvalues of the matrices $B_k^{(0)} = \frac{y_{k,l}^T y_{k,l}}{s_{k,l}^T y_{k,l}} I$ at the start of the sampled LBFGS update cycles are bounded above and away from zero, for all k and l. We now use a Trace-Determinant argument to show that the eigenvalues of B_k are bounded above and away from zero.

Let $Tr(B)$ and $\det(B)$ denote the trace and determinant of matrix B, respectively. The trace of the matrix B_{k+1} can be expressed as,

$$\begin{aligned} Tr(B_{k+1}) &= Tr(B_k^{(0)}) - Tr\sum_{i=1}^{\tilde{m}_k}\left(\frac{B_k^{(i-1)} s_{k,i} s_{k,i}^T B_k^{(i-1)}}{s_{k,i}^T B_k^{(i-1)} s_{k,i}}\right) + Tr\sum_{i=1}^{\tilde{m}_k}\frac{y_{k,i} y_{k,i}^T}{y_{k,i}^T s_{k,i}} \\ &\leq Tr(B_k^{(0)}) + \sum_{i=1}^{\tilde{m}_k}\frac{\|y_{k,i}\|^2}{y_{k,i}^T s_{k,i}} \\ &\leq Tr(B_k^{(0)}) + \tilde{m}_k L \\ &\leq Tr(B_k^{(0)}) + mL \leq C_1, \end{aligned} \tag{A.10}$$

for some positive constant C_1, where the inequalities above are due to (A.6), the fact that the eigenvalues of the initial L-BFGS matrix $B_k^{(0)}$ are bounded above and away from zero, and the fact that $\tilde{m}_k \leq m$ for all k.

Using a result due to Powell [46], the determinant of the matrix B_{k+1} generated by the sampled LBFGS method can be expressed as,

$$\begin{aligned}
\det(B_{k+1}) &= \det(B_k^{(0)}) \prod_{i=1}^{\tilde{m}_k} \frac{y_{k,i}^T s_{k,i}}{s_{k,i}^T B_k^{(i-1)} s_{k,i}} \\
&= \det(B_k^{(0)}) \prod_{i=1}^{\tilde{m}_k} \frac{y_{k,i}^T s_{k,i}}{s_{k,i}^T s_{k,i}} \frac{s_{k,i}^T s_{k,i}}{s_{k,i}^T B_k^{(i-1)} s_{k,i}} \\
&\geq \det(B_k^{(0)}) \Big(\frac{\mu}{C_1}\Big)^{\tilde{m}_k} \\
&\geq \det(B_k^{(0)}) \Big(\frac{\mu}{C_1}\Big)^{m} \geq C_2, \qquad \text{(A.11)}
\end{aligned}$$

for some positive constant C_2, where the above inequalities are due to the fact that the largest eigenvalue of $B_k^{(i)}$ is less than C_1 and Assumption 5.2, and the fact that $\frac{\mu}{C_1} < 1$.

The trace (A.10) and determinant (A.11) inequalities derived above imply that largest eigenvalues of all matrices B_k are bounded above, uniformly, and that the smallest eigenvalues of all matrices B_k are bounded away from zero, uniformly. □

A.3 Proof of Theorem 5.4

Theorem 5.4. *Suppose that Assumptions 5.1 and 5.2 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Let $\{w_k\}$ be the iterates generated by Algorithm 2, where*

$$0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L},$$

and w_0 is the starting point. Then for all $k \geq 0$,

$$F(w_k) - F^\star \leq \big(1 - \alpha\mu\mu_1\big)^k \left[F(w_0) - F^\star\right].$$

Proof. We have that

$$\begin{aligned}
F(w_{k+1}) &= F(w_k - \alpha H_k \nabla F(w_k)) \\
&\leq F(w_k) + \nabla F(w_k)^T(-\alpha H_k \nabla F(w_k)) + \frac{L}{2}\|\alpha H_k \nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F(w_k) + \frac{\alpha^2 \mu_2^2 L}{2}\|\nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\mu_1 \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 L}{2}\|\nabla F(w_k)\|^2 \\
&= F(w_k) - \alpha\left(\mu_1 - \alpha\frac{\mu_2^2 L}{2}\right)\|\nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2, \qquad \text{(A.12)}
\end{aligned}$$

where the first inequality is due to Assumption 5.2, the second and third inequalities arise as a consequence of Lemma 5.3 and the last inequality is due to the choice of the steplength. By strong convexity, we have $2\mu(F(w) - F^\star) \leq \|\nabla F(w)\|^2$, and thus

$$F(w_{k+1}) \leq F(w_k) - \alpha\mu\mu_1(F(w_k) - F^\star).$$

Subtracting $F^\star$ from both sides,

$$F(w_{k+1}) - F^\star \leq (1 - \alpha\mu\mu_1)(F(w_k) - F^\star).$$

Recursive application of the above inequality yields the desired result. □

A.4 Proof of Lemma 5.7

Lemma 5.7. *Suppose that Assumptions 5.1 and 5.6 hold. Let $\{H_k\}$ be the inverse Hessian approximations generated by Algorithm 2, with the modification that the inverse approximation update is performed using only the curvature pairs that satisfy (5.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (5.1). Then, there exist constants $0 < \mu_1 \leq \mu_2$ such that*

$$\mu_1 I \preceq H_k \preceq \mu_2 I, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{A.13}$$

Proof. As in the proof of Lemma 5.3, note that there is a chance that no curvature pairs are selected in Algorithm 1. In this case, the inverse Hessian approximation is $H_k = I$, and thus $\mu_1 = \mu_2 = 1$ and condition (A.13) is satisfied.

Similar to the proof of Lemma 5.3, we study the direct Hessian approximation $B_k = H_k^{-1}$. In our algorithm, there are two options for updating the curvature pairs $s_{k,j}$ and $y_{k,j}$:

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla F(w) - \nabla F(\bar{w}_j) \qquad \text{Option I}, \tag{A.14}$$

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla^2 F(w_k) s_k \qquad \text{Option II}, \tag{A.15}$$

for $j = 1, \ldots, m$. Let $\tilde{m}_k \in \{1, ..., m\}$ denote the number of curvature pairs that satisfy (5.1) at the kth iteration, where m is the memory. At the kth iteration, given a set of curvature pairs $(s_{k,j}, y_{k,j})$, for $j = 1, \ldots, \tilde{m}_k$ we update the Hessian approximation recursively (using the procedure described in the proof of Lemma 5.3, and set $B_{k+1} = B_k^{\tilde{m}_k}$.

In this setting, the skipping mechanism (5.1) provides both an upper and lower bound on the quantity $\frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}}$, for both Options, which in turn ensures that the initial sampled LBFGS Hessian approximation is bounded above and away from zero.

The lower bound is attained by repeated application of Cauchy's inequality to condition (5.1). We have from (5.1) that

$$\epsilon \|s_{k,j}\|^2 \leq y_{k,j}^T s_{k,j} \leq \|y_{k,j}\| \|s_{k,j}\| \quad \Rightarrow \quad \|s_{k,j}\| \leq \frac{1}{\epsilon} \|y_{k,j}\|.$$

It follows that

$$s_{k,j}^T y_{k,j} \leq \|s_{k,j}\| \|y_{k,j}\| \leq \frac{1}{\epsilon} \|y_{k,j}\|^2 \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{s_{k,j}^T y_{k,j}} \geq \epsilon. \tag{A.16}$$

The upper bound is attained by the Lipschitz continuity of gradients,

$$\begin{aligned} y_{k,j}^T s_{k,j} &\geq \epsilon \|s_{k,j}\|^2 \\ &\geq \epsilon \frac{\|y_{k,j}\|^2}{L} \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{s_{k,j}^T y_{k,j}} \leq \frac{L^2}{\epsilon}. \end{aligned} \tag{A.17}$$

Combining (A.16) and (A.17), we have

$$\epsilon \leq \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \leq \frac{L^2}{\epsilon}.$$

The bounds on $\frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}}$ prove that for any l chosen uniformly at random from $\{1, \ldots, \tilde{m}_k\}$ the eigenvalues of the matrices $B_k^{(0)} = \frac{y_{k,l}^T y_{k,l}}{s_{k,l}^T y_{k,l}} I$ at the start of the sampled LBFGS update cycles are bounded above and away from zero, for all k and l. The rest of the proof follows the same trace-determinant argument as in the proof of Lemma 5.3, the only difference being that the last inequality in A.11 comes as a result of the cautious update strategy. □

A.5 Proof of Theorem 5.8

Theorem 5.8. *Suppose that Assumptions 5.1, 5.5 and 5.6 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 2, with the modification that the inverse Hessian approximation update is performed*

using only the curvature pairs that satisfy (5.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (5.1), where

$$0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L},$$

and w_0 is the starting point. Then,

$$\lim_{k\to\infty} \|\nabla F(w_k)\| \to 0, \tag{A.18}$$

and, moreover, for any $\tau > 1$,

$$\frac{1}{\tau}\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1\tau} \xrightarrow{\tau\to\infty} 0.$$

Proof. We start with (A.12)

$$F(w_{k+1}) \leq F(w_k) - \alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2.$$

Summing both sides of the above inequality from $k = 0$ to $\tau - 1$,

$$\sum_{k=0}^{\tau-1}(F(w_{k+1}) - F(w_k)) \leq -\sum_{k=0}^{\tau-1}\alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2.$$

The left-hand-side of the above inequality is a telescoping sum and thus,

$$\sum_{k=0}^{\tau-1}[F(w_{k+1}) - F(w_k)] = F(w_\tau) - F(w_0) \geq \widehat{F} - F(w_0),$$

where the inequality is due to $\hat{F} \leq F(w_\tau)$ (Assumption 5.5). Using the above, we have

$$\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1}. \tag{A.19}$$

Taking limits we obtain,

$$\lim_{\tau\to\infty}\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 < \infty,$$

which implies (A.18). Dividing (A.19) by τ we conclude

$$\frac{1}{\tau}\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1\tau}.$$

□

A.6 Proof of Lemma 5.10

Lemma 5.10. *Suppose that Assumptions 5.1, 5.6 and 5.9 hold. Let $\{B_k\}$ be the Hessian approximations generated by Algorithm 3, with the modification that the approximation update is performed using only the curvature pairs that satisfy (5.2), for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (5.2). Then, there exists a constant $\nu_2 > 0$ such that*

$$\|B_k\| \leq \nu_2, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{A.20}$$

Proof. As in the proof of Lemma 5.3, note that there is a chance that no curvature pairs are selected in Algorithm 1. In this case, the Hessian approximation is $B_k = I$, and thus $\nu_2 = 1$ and condition (A.20) is satisfied.

We now consider the case where at least one curvature pair is selected by Algorithm 1. In this case, the sampled LSR1 updating formula is given as follows. Let $\tilde{m}_k \in \{1, ..., m\}$ denote the number of curvature pairs that satisfy (5.2) at the kth iteration, where m is the memory. At the kth iteration, given a set of curvature pairs $(s_{k,j}, y_{k,j})$, for $j = 1, \ldots, \tilde{m}_k$

1. Set $B_k^{(0)} = \gamma_k I$, where $0 \leq \gamma_k < \gamma$.
2. For $i = 1, \ldots, \tilde{m}_k$ compute

$$B_k^{(i)} = B_k^{(i-1)} + \frac{(y_{k,i} - B_k^{(i-1)} s_{k,i})(y_{k,i} - B_k^{(i-1)} s_{k,i})^T}{(y_{k,i} - B_k^{(i-1)} s_{k,i})^T s_{k,i}}.$$

3. Set $B_{k+1} = B_k^{(\tilde{m}_k)}$.

In our algorithm (Algorithm 1), there are two options for constructing the curvature pairs $s_{k,j}$ and $y_{k,j}$. At the current iterate w_k we sample points $\bar{w}_j$ for $j = 1, \ldots, m$ and set

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla F(w_k) - \nabla F(\bar{w}_j) \qquad \text{Option I,} \tag{A.21}$$

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla^2 F(w_k) s_k \qquad \text{Option II.} \tag{A.22}$$

Given a set of $\tilde{m}_k$ curvature pairs that satisfy (5.2), we now prove an upper bound for $\|B_k\|$. We first prove the bound for a given iteration k and for all updates to the Hessian approximation $i = 0, 1, \ldots, \tilde{m}_k$ ($\|B_k^i\|$), and then get an upper bound for all k ($\|B_k\|$).

For a given iteration k, we prove a bound on $\|B_k^i\|$ via induction, and show

$$\|B_k^{(i)}\| \leq \left(1 + \frac{1}{\epsilon}\right)^i \gamma_k + \left[\left(1 + \frac{1}{\epsilon}\right)^i - 1\right] \bar{\gamma}_k. \tag{A.23}$$

For $i = 0$, the bound holds trivially since $B_k^{(0)} = \gamma_k I$. Now assume that (A.23) holds true for some $i \geq 0$. Note that all the curvature pairs that are used in the update of the Hessian approximation satisfy (5.2). By the definition of the SR1 updates, we have for some index $i + 1$ that

$$B_k^{(i+1)} = B_k^{(i)} + \frac{(y_{k,i+1} - B_k^{(i)} s_{k,i+1})(y_{k,i+1} - B_k^{(i)} s_{k,i+1})^T}{(y_{k,i+1} - B_k^{(i)} s_{k,i+1})^T s_{k,i+1}},$$

and thus

$$\begin{aligned}
\|B_k^{(i+1)}\| &\leq \|B_k^{(i)}\| + \left\| \frac{(y_{k,i+1} - B_k^{(i)} s_{k,i+1})(y_{k,i+1} - B_k^{(i)} s_{k,i+1})^T}{(y_{k,i+1} - B_k^{(i)} s_{k,i+1})^T s_{k,i+1}} \right\|, \\
&\leq \|B_k^{(i)}\| + \frac{\|(y_{k,i+1} - B_k^{(i)} s_{k,i+1})(y_{k,i+1} - B_k^{(i)} s_{k,i+1})^T\|}{\epsilon \|y_{k,i+1} - B_k^{(i)} s_{k,i+1}\| \|s_{k,i+1}\|} \\
&\leq \|B_k^{(i)}\| + \frac{\|y_{k,i+1} - B_k^{(i)} s_{k,i+1}\|}{\epsilon \|s_{k,i+1}\|} \\
&\leq \|B_k^{(i)}\| + \frac{\|y_{k,i+1}\|}{\epsilon \|s_{k,i+1}\|} + \frac{\|B_k^{(i)} s_{k,i+1}\|}{\epsilon \|s_{k,i+1}\|} \\
&\leq \|B_k^{(i)}\| + \frac{\|y_{k,i+1}\|}{\epsilon \|s_{k,i+1}\|} + \frac{\|B_k^{(i)}\|}{\epsilon} \\
&= \left(1 + \frac{1}{\epsilon}\right) \|B_k^{(i)}\| + \frac{\bar{\gamma}_k}{\epsilon}
\end{aligned}$$

where the first inequality is due to the application of the triangle inequality, the second inequality is due to condition (5.2), the fourth inequality is due to the application of the triangle inequality, and the fifth inequality is due to application of Cauchy's inequality and in the last inequality we used that $\bar{\gamma}_k \geq \bar{\gamma}_{k,i+1} = \frac{\|y_{k,i+1}\|}{\|s_{k,i+1}\|} > 0$. Substituting (A.23),

$$\begin{aligned}
\|B_k^{(i+1)}\| &\leq \left(1 + \frac{1}{\epsilon}\right) \left[\left(1 + \frac{1}{\epsilon}\right)^i \gamma_k + \left[\left(1 + \frac{1}{\epsilon}\right)^i - 1\right] \bar{\gamma}_k\right] + \frac{\bar{\gamma}_k}{\epsilon} \\
&= \left(1 + \frac{1}{\epsilon}\right)^{i+1} \gamma_k + \left[\left(1 + \frac{1}{\epsilon}\right)^{i+1} - 1\right] \bar{\gamma}_k
\end{aligned}$$

which completes the inductive proof. Thus, for any k we have an upper bound on the Hessian approximation. Therefore, since $B_{k+1} = B_k^{(\tilde{m}_k)}$, the sampled SR1 Hessian approximation constructed at the kth iteration satisfies

$$\|B_{k+1}\| \leq \left(1 + \frac{1}{\epsilon}\right)^{i+1} \gamma_k + \left[\left(1 + \frac{1}{\epsilon}\right)^{i+1} - 1\right] \bar{\gamma}_k.$$

Now we generalize the result for all iterations k. For $k = 0$, the bound holds trivially, since the first step of the sampled LSR1 method is a gradient method ($B_0 = I$). For $k \geq 1$, we assume that $\gamma_k \leq \gamma < \infty$ and $\bar{\gamma}_k \leq \bar{\gamma} < \infty$ for all k, and thus

$$\begin{aligned}\|B_{k+1}\| &\leq \left(1 + \frac{1}{\epsilon}\right)^{i+1} \gamma_k + \left[\left(1 + \frac{1}{\epsilon}\right)^{i+1} - 1\right] \bar{\gamma}_k \\ &\leq \left(1 + \frac{1}{\epsilon}\right)^{i+1} \gamma + \left[\left(1 + \frac{1}{\epsilon}\right)^{i+1} - 1\right] \bar{\gamma} = \nu_2,\end{aligned}$$

for some $\nu_2 > 0$. This completes the proof.

□

A.7 Proof of Theorem 5.11

Theorem 5.11. *Suppose that Assumptions 5.1, 5.5, 5.6 and 5.9 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 3, with the modification that the Hessian approximation update is performed using only the curvature pairs that satisfy 5.2, for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (5.2). Then,*

$$\lim_{k \to \infty} \|\nabla F(w_k)\| = 0.$$

Proof. Assume, for the purpose of a establishing contradiction, that there is a subsequence of successful iterations (where $\rho_k > \eta_1$, Line 6, Algorithm 3), indexed by $t_i \subseteq \mathcal{S}$ where $\mathcal{S} = \{k \geq 0 | \rho_k \geq \eta_1\}$, such that

$$\|\nabla F(w_{t_i})\| \geq 2\delta > 0 \tag{A.24}$$

for some $\epsilon > 0$ and for all i. Theorem 6.4.5 from [14] then ensures the existence for each t_i of a first successful iteration $\ell(t_i) > t_i$ such that

$$\|\nabla F(w_{\ell(t_i)})\| < \delta > 0.$$

Let $\ell_i = \ell(t_i)$, we thus obtain that there is anotehr subsequence of $\mathcal{S}$ indexed by $\{\ell_i\}$ such that

$$\|\nabla F(w_k)\| \geq \delta, \quad \text{for} \quad t_i \leq k < \ell_i \quad \text{and} \quad \|\nabla F(w_{\ell_i})\| < \delta. \tag{A.25}$$

We now restrict our attention to the subsequence of successful iterations whose indices are in the set

$$\mathcal{K} = \{k \in \mathcal{S} | t_i \leq k < \ell_i\},$$

where t_i and ℓ_i belong to the subsequences $\mathcal{S}$ and $\mathcal{K}$, respectively.

Using Assumption 5.9, the fact that $\mathcal{K} \subseteq \mathcal{S}$ and (A.25), we deduce that for $k \in \mathcal{K}$

$$F(w_k) - F(w_k) \geq \eta_1 [m_k(0) - m_k(p_k)] \geq \xi \delta \eta_1 \min\left[\frac{\delta}{\nu_2 + 1}, \Delta_k\right] \tag{A.26}$$

where we used the result of Lemma 5.10. Since the sequence $\{F(w_k)\}$ is monotonically decreasing and bounded below (Assumption 5.5), it is convergent, and the left-hand-side of (A.26) must tend to zero as $k \to \infty$. Thus,

$$\lim_{k \to \infty,\ k \in \mathcal{K}} \Delta_k = 0. \tag{A.27}$$

As a consequence, the term containing Δ_k is the dominant term in the $\min$ (A.26) and we have, for $k \in \mathcal{K}$ sufficiently large,

$$\Delta_k \leq \frac{F(w_k) - F(w_{k+1})}{(\nu_2 + 1)\delta\eta_1}. \tag{A.28}$$

From this bound, we deduce that, for i sufficiently large

$$\|w_{t_i} - w_{\ell_i}\| \leq \sum_{j=t_i,\, j\in\mathcal{K}}^{\ell_i - 1} \|w_j - w_{j+1}\| \leq \sum_{j=t_i,\, j\in\mathcal{K}}^{\ell_i - 1} \Delta_j \leq \frac{F(w_{t_i}) - F(w_{\ell_i})}{(\nu_2 + 1)\delta\eta_1}. \tag{A.29}$$

As a consequence of Assumption 5.5 and the monotonicity of the sequence $\{F(w_k)\}$, we have that the right-hand-side of (A.29) must converge to zero, and thus $\|w_{t_i} - w_{\ell_i}\| \to 0$ as $i \to \infty$.

By continuity of the gradient (Assumption 5.1), we thus deduce that $\|\nabla F(w_{t_i}) - \nabla F(w_{\ell_i})\| \to 0$. However, this is impossible because of the definitions of $\{t_i\}$ and $\{\ell_i\}$, which imply that $\|\nabla F(w_{t_i}) - \nabla F(w_{\ell_i})\| \geq \delta$. Hence, no subsequence satisfying (A.24) can exist, and the theorem is proved. □

B Additional Numerical Experiments and Method Details

In this section, we present additional numerical results and expand on some details about the methods.[4]

B.1 Motivation Figure

In this section, we present more motivating plots showing the accuracy vs. iterations and accuracy vs. epochs for a toy classification problem. In the following experiments, we ran each method from 10 different initial points.

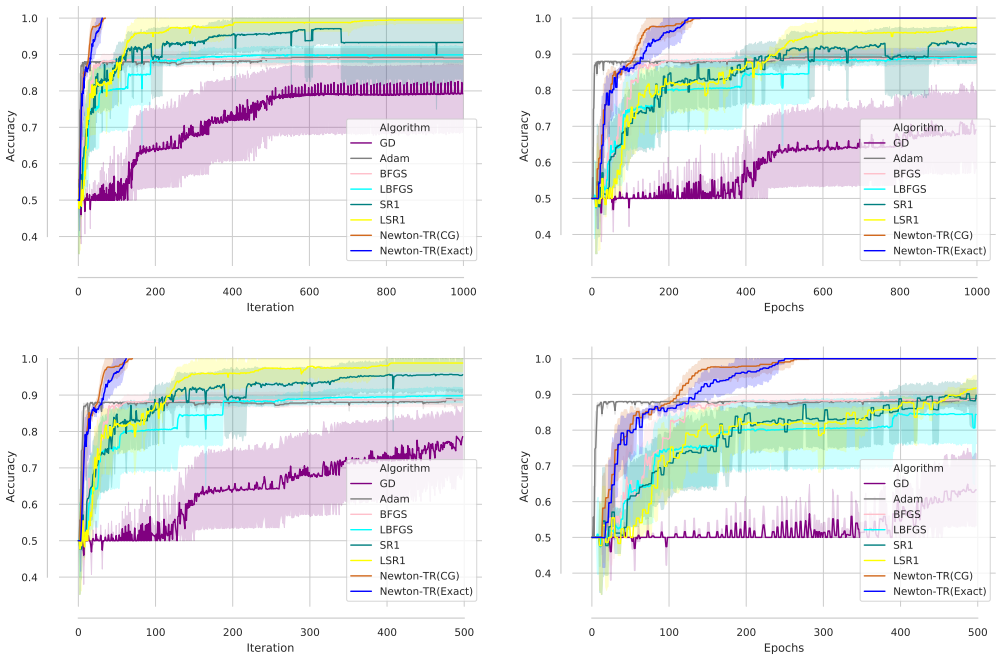

Figure 7: Comparison of Gradient Descent (GD), ADAM, BFGS, LBFGS, SR1, LSR1, Newton-TR (Exact), Newton-TR (CG) on a toy classification problem in terms of iterations and epochs.

[4] All experiments we run on a machine with the following specifications: 24 cores Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz; 128 GB RAM; 2 K80 GPUs; Linux Debian GNU/Linux 8.10 (jessie); TensorFlow 1.12.2; CUDA 8.0; Python 2.7.

B.2 Eigenvalue Figures

In this section, we describe the procedure in which Figure 2 was constructed. We plot the same figure below for ease of exposition, and also plot a similar figure for another network.

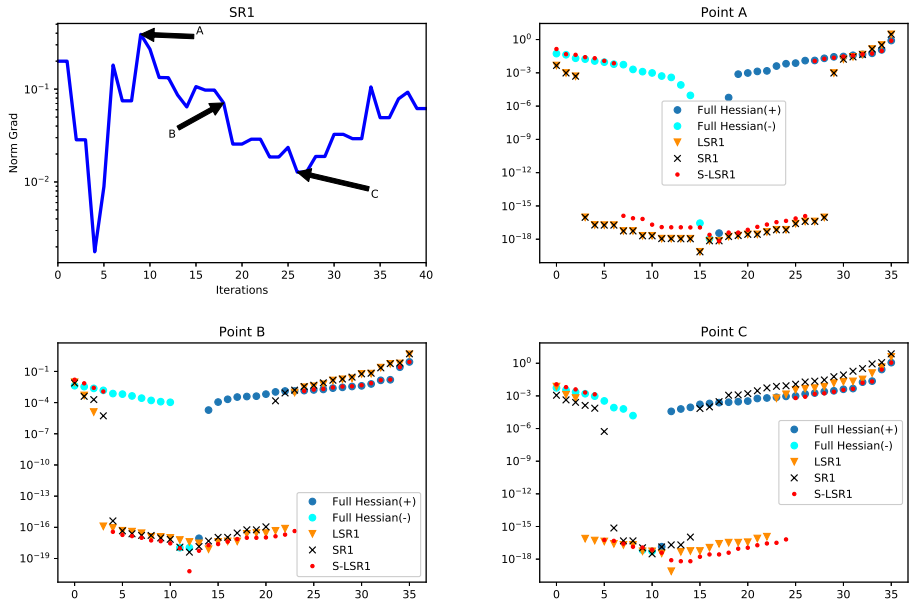

Figure 8: Comparison of the eigenvalues of SR1, LSR1 and S-LSR1 at different points for a toy classification problem.

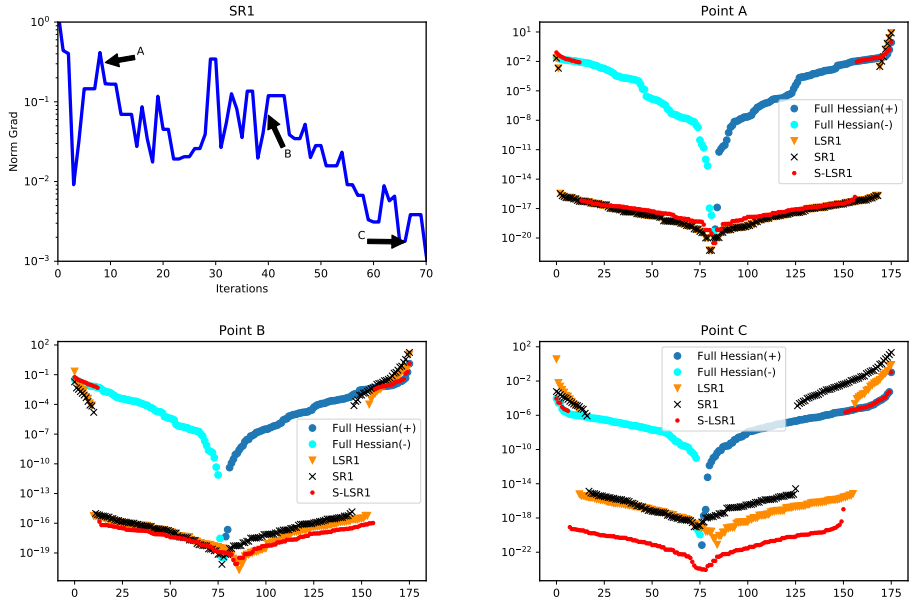

Figure 9: Comparison of the eigenvalues of SR1, LSR1 and S-LSR1 at different points for a toy classification problem.

To calculate the eigenvalues for SR1, LSR1 and S-LSR1 we used the following procedure.

1. We ran the SR1 method for T iterations on a toy classification problem. During the optimization, we computed the eigenvalues of the SR1 Hessian approximation at several points (e.g., A, B and C); black $\times$ marks on plots.
2. We stored all the curvature pairs $\{s_k, y_k\}_{k=1}^{T}$ and the iterates $\{w_k\}_{k=1}^{T}$.
3. We constructed the true Hessian at all iterations and computed the eigenvalues of the true Hessian; dark blue $\bullet$ (positive eigenvalues) and light blue $\bullet$ (negative eigenvalues) marks on plots.
4. We then computed the limited-memory SR1 Hessian approximations at several points (e.g., A, B and C) using the m most recent pairs and computed the eigenvalues of the approximations; orange $\blacktriangledown$ marks on plots.
5. Finally, we used the iterate information at points A, B and C, sampled m points at random around those iterates with sampling radius r, constructed the sampled LSR1 Hessian approximations and computed the eigenvalues of the approximations; red $\bullet$ marks on plots.

Note: for Figure 8 we used $T = 40$, $m = 16$ and $r = 0.01$, and for Figure 9 we used $T = 70$, $m = 32$ and $r = 0.01$.

As is clear, the eigenvalues of the sampled LSR1 Hessian approximations better match the eigenvalues of the true Hessian. Similar results were obtained for other problems and for different parameters m and r.

B.3 Trust-Region Management Subroutine

In this section we present, in detail, the Trust-Region management subroutine ($\Delta_{k+1} = \texttt{adjustTR}(\Delta_k, \rho_k)$) that is used in Algorithm 3. See [45] for further details.

Algorithm 4 $\Delta_{k+1} = \texttt{adjustTR}(\Delta_k, \rho_k, \eta_2, \eta_3, \gamma_1, \zeta_1, \zeta_2)$: Trust-Region management subroutine

Input: Δ_k (current trust region radius), $0 \leq \eta_3 < \eta_2 < 1$, $\gamma_1 \in (0, 1)$, $\zeta_1 > 1$, $\zeta_2 \in (0, 1)$ (trust region parameters).

if $\rho_k > \eta_2$ **then**
    if $\|p_k\| \leq \gamma_1 \Delta_k$ **then**
        Set $\Delta_{k+1} = \Delta_k$
    else
        Set $\Delta_{k+1} = \zeta_1 \Delta_k$
    end if
else if $\eta_3 \leq \rho_k \leq \eta_2$ **then**
    Set $\Delta_{k+1} = \Delta_k$
else
    $\Delta_{k+1} = \zeta_2 \Delta_k$
end if

B.4 Hessian-Free Implementation of Limited-Memory SR1 Methods

In this section, we discuss the practical implementation of limited-memory SR1 methods where we need not construct the Hessian approximation B_k explicitly. For the purpose of this discuss we focus on the S-LSR1 method, but a similar approach can be used for the LSR1 method too. To do so, we utilize the compact representation of the Hessian approximation discussed in [12] which is equivalent to B_k in (2.5). The compact representation can be expressed as follows:

$$B_{k+1} = B_k^{(0)} + (Y_k - B_k^{(0)} S_k)\Big(D_k + L_k + L_k^T - S_k^T B_k^{(0)} S_k\Big)^{-1}(Y_k - B_k^{(0)} S_k)^T, \tag{B.1}$$

where $S_k = [s_{k,1}, s_{k,2}, \ldots, s_{k,m}] \in \mathbb{R}^{d\times m}$ and $Y_k = [y_{k,1}, y_{k,2}, \ldots, y_{k,m}] \in \mathbb{R}^{d\times m}$, and $B_k^{(0)}$ is a symmetric positive definite initial Hessian approximation, which for the purpose of this discussion we assume has the form $B_k^{(0)} = \gamma_k I$ ($0 \leq \gamma_k < \gamma < \infty$). In (B.1), D_k and L_k are two $m \times m$ matrices that are defined as follows,

$$D_k = diag[s_{k,1}^T y_{k,1}, \ldots, s_{k,m}^T y_{k,m}] \tag{B.2}$$

$$(L_k)_{i,j} = \begin{cases} s_{k,i-1}^T y_{k,j-1} & \text{if i > j} \\ 0 & \text{otherwise} \end{cases} \tag{B.3}$$

The curvature pairs in the matrices S_k and Y_k are pairs that satisfy the condition given in (5.2).

In large-scale applications, it is not memory-efficient, or even possible for some applications, to store a $d \times d$ Hessian approximation matrix B_{k+1}. Instead, we can calculate the Hessian vector product $B_{k+1}v$, for some $v \in \mathbb{R}^d$, by leveraging the compact form of B_k in (B.1) as follows:

$$B_{k+1}v = B_k^{(0)} v + (Y_k - B_k^{(0)} S_k)\Big(D_k + L_k + L_k^T - S_k^T B_k^{(0)} S_k\Big)^{-1}(Y_k - B_k^{(0)} S_k)^T v \tag{B.4}$$

The above, $B_{k+1}v$, is very efficient in terms of memory, and even more importantly efficient to compute; the complexity of computing $B_{k+1}v$ is $\mathcal{O}(m^2 d)$.

In Algorithm 3, we need to compute and use $B_{k+1}v$ in the following parts: (1) in checking the condition (5.2); (2) as part of the computation of solving the subproblem (2.4) using the CG solver (see [45]); and, (3) in the calculation of ρ_k.

In the remainder of this section we describe the steps for checking whether the curvature pairs constructed by Algorithm 1 satisfy (5.2). This is by no means a trivial task; several researchers have proposed mechanisms for doing this [8, 38] by using spectral decomositions of B_k. We propose to do this in a dynamic manner leveraging (B.4).

The condition that we want to check (5.2) has the following form:

$$|s_{k,i}^T (y_{k,i} - B_k^{(i-1)} s_{k,i})| \geq \epsilon \|s_{k,i}\| \|y_{k,i} - B_k^{(i-1)} s_{k,i}\|, \tag{B.5}$$

for $i = 1, \ldots, m$, where $B_k^{(i)}$ are constructed recursively via,

$$B_k^{(i)} = B_k^{(0)} + (Y_k^i - B_k^{(0)} S_k^i)\Big(D_k^i + L_k^i + (L_k^i)^T - (S_k^i)^T B_k^{(0)} S_k^i\Big)^{-1}(Y_k^i - B_k^{(0)} S_k^i)^T, \tag{B.6}$$

$S_k^i = [s_{k,1}, s_{k,2}, \ldots, s_{k,i}]$, $Y_k^i = [y_{k,1}, y_{k,2}, \ldots, y_{k,i}]$, and D_k^i and L_k^i are defined in equations (B.2) and (B.3), respectively, using S_k^i and Y_k^i. Of course we want to check condition (B.5) without explicitly forming the matrices $B_k^{(i)}$, and instead construct $B_k^{(i)} s_{k,i+1}$ directly. To this end, by using (B.4), for any $i = 0, \ldots, m-1$ we have:

$$B_k^{(i)} s_{k,i+1} = B_k^{(0)} s_{k,i+1} + (Y_k^i - B_k^{(0)} S_k^i)\Big(D_k^i + L_k^i + (L_k^i)^T - (S_k^i)^T B_k^{(0)} S_k^i\Big)^{-1}(Y_k^i - B_k^{(0)} S_k^i)^T s_{k,i+1} \tag{B.7}$$

where the matrices S_k^i, Y_k^i, D_k^i and L_k^i are defined as above.

The steps for checking (5.2) are as follows:

1. Consider the kth iteration of Algorithm 3, where the pairs $\bar{S}_k = [s_{k,1}, \ldots, s_{k,m}]$ and $\bar{Y}_k = [y_{k,1}, \ldots, y_{k,m}]$ and B_k^0 are constructed by Algorithm 1. Let $S_k = [\,]$ and $Y_k = [\,]$ be two empty matrices.
2. For any $i = 1, \ldots, m$, consider the pair $(s_{k,i}, y_{k,i})$ and compute $B_k^{(i-1)} s_{k,i}$ by using (B.7) and the updated lists S_k and Y_k. Note that for $i = 1$ the matrices S_k and Y_k are empty and $B_k^{(0)}$ is the initial Hessian approximation, thus condition (B.5) can be checked directly.
   (a) If condition (B.5) is satisfied for this pair, then $S_k = [S_k \; s_{k,i}]$ add $Y_k = [Y_k \; y_{k,i}]$
   (b) Else, discard the pair $(s_{k,i}, y_{k,i})$

Using the mechanism described above, we recursively check condition (B.5), and construct well defined pairs S_k and Y_k which are used for the calculation of p_k and ρ_k. As mentioned above, we can use the same idea for implementing the LSR1 method.

B.5 Implementation Details

In this section, we discuss the implementation details for all the methods.[5]

- For **ADAM**, we tuned the steplength and batch size for each problem independently. We used a batch size of 1.
- For **GD** and **BFGS**-type methods, we computed the steplength using a backtracking Armijo line search [45].
- For **SR1**-type methods, we solved the trust-region subproblems (2.4) using CG-Steihaug [45].
- For **BFGS** and **SR1**, we constructed the full (inverse) Hessian approximations explicitly, whereas for the limited-memory we never constructed the full matrices.
- For **limited-memory BFGS** methods we used the two-loop recursion to compute the search direction [45].
- Implementing the **limited memory SR1** methods is not trivial; we made use of the compact representations of the SR1 matrices [12] and computed the steps dynamically; see Appendix B.4 for details.

[5] All codes to reproduce the results presented in this section are available at: http://github.com/ANONYMOUS/LINK. The code will be released upon acceptance of the paper.

B.6 Toy Example

In this section, we present additional numerical results for the toy classification problem described in Section 6. In the following experiments, we ran each method from 100 different initial points. The details of the three networks are summarized in Table 1.

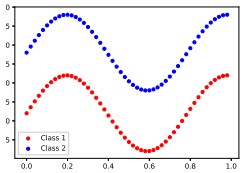

Figure 10: Toy Classification Problem

Network	Structure	d
`small`	2-2-2-2-2-2	36
`medium`	2-4-8-8-4-2	176
`large`	2-10-20-20-10-2	908

Table 1: Toy Classification Problem: Neural Network Details

B.6.1 Performance of Methods on small, medium and large toy classification problems - Box-plots

The following box-plots show the accuracy achieved by different methods for different budgets (epochs) and iterations.

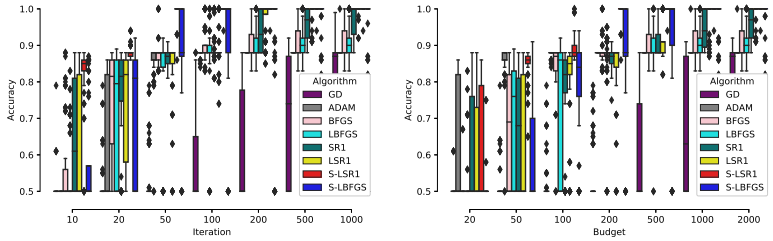

Figure 11: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (small network).

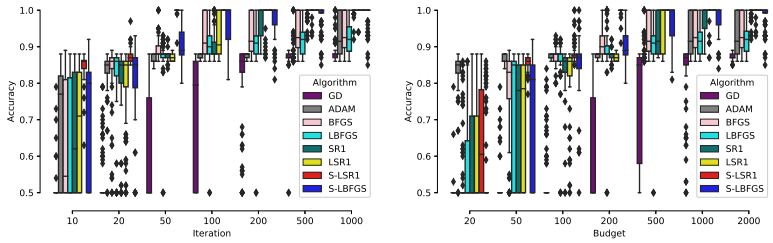

Figure 12: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (medium network).

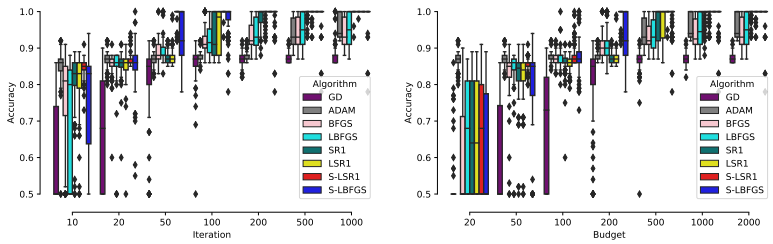

Figure 13: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (large network).

B.6.2 Performance of Methods on small, medium and large toy classification problems

In this section, we present more experiments showing accuracy vs. iterations and accuracy vs. epochs for different methods on toy classification problem.

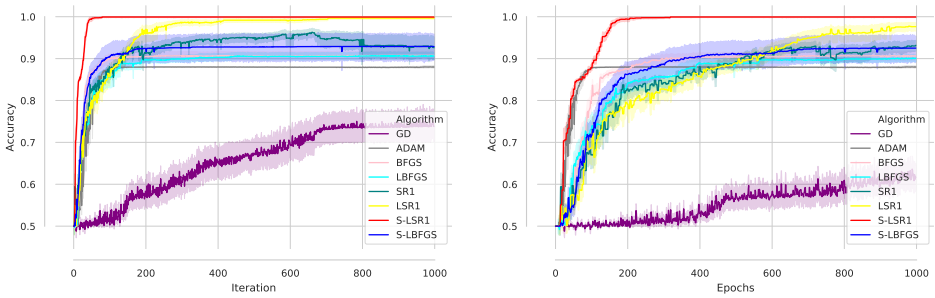

Figure 14: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (small network).

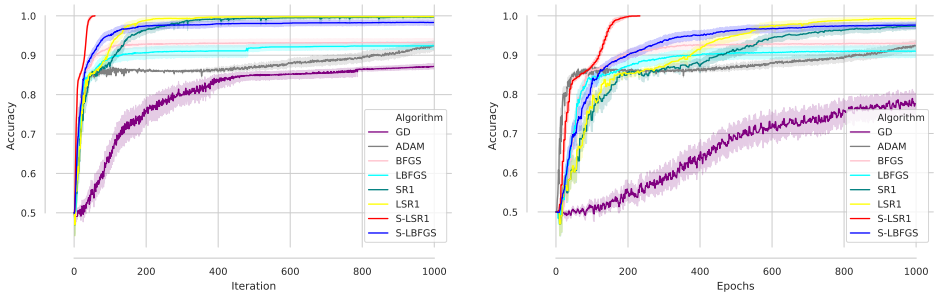

Figure 15: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (medium network).

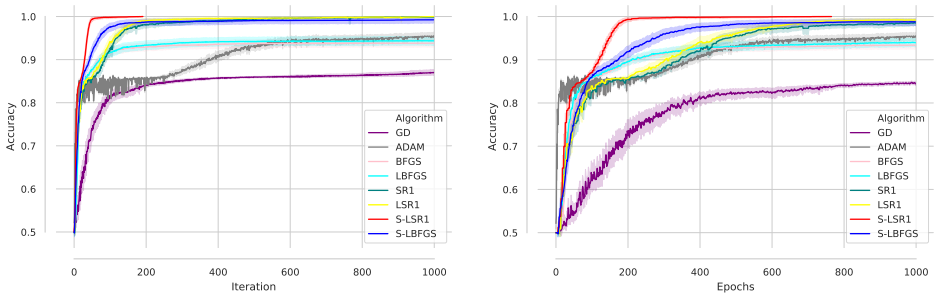

Figure 16: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (large network).

B.6.3 Comparison of BFGS-type methods

In this section, we present more experiments showing the accuracy achieved in terms of iterations and epochs for BFGS-type methods on toy classification problem.

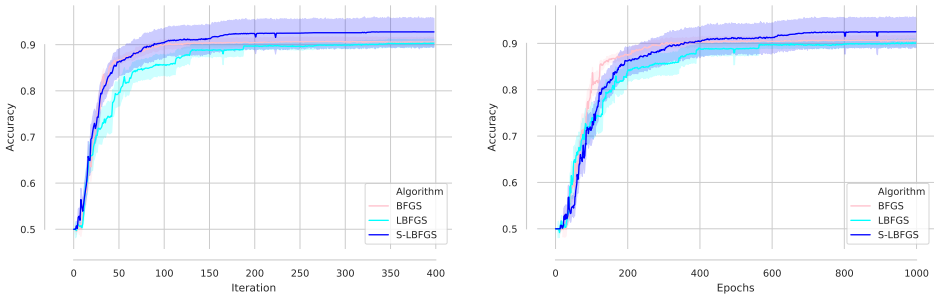

Figure 17: Performance of BFGS-type methods on toy classification problem (small network).

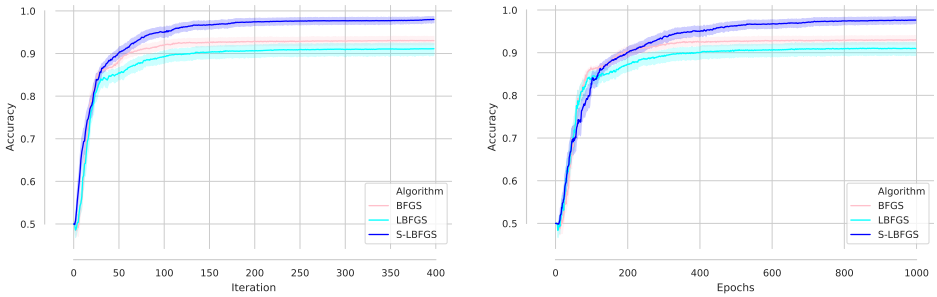

Figure 18: Performance of BFGS-type methods on toy classification problem (medium network).

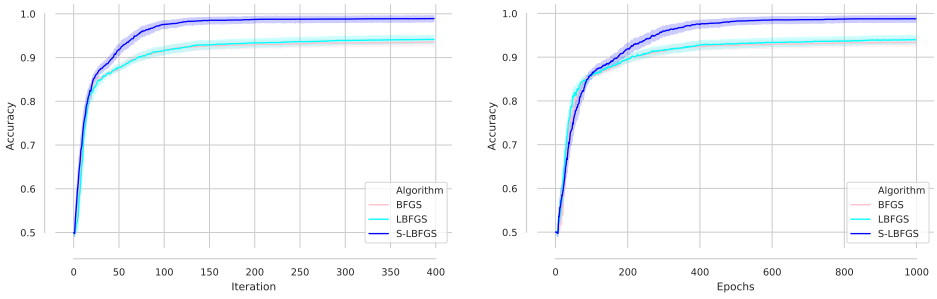

Figure 19: Performance of BFGS-type methods on toy classification problem (large network).

B.6.4 Comparison of SR1-type methods

In this section, we present more experiments showing the accuracy achieved in terms of iterations and epochs for SR1-type methods on toy classification problem.

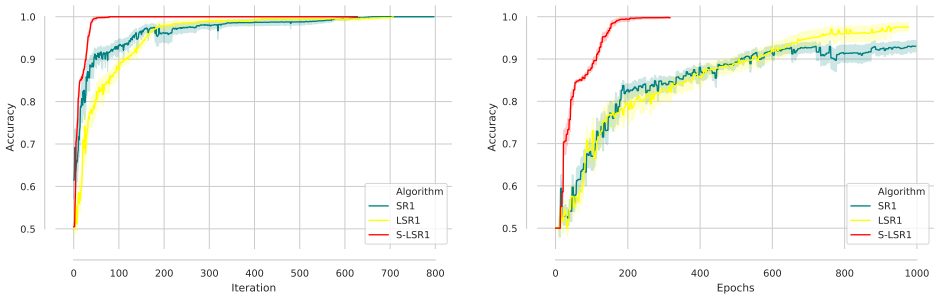

Figure 20: Performance of SR1-type methods on toy classification problem (small network).

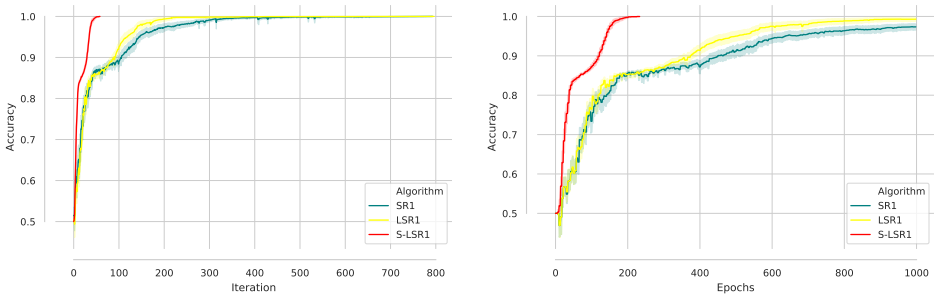

Figure 21: Performance of SR1-type methods on toy classification problem (medium network).

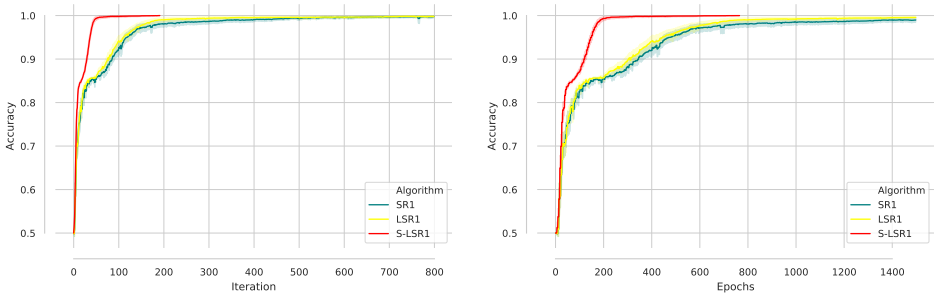

Figure 22: Performance of SR1-type methods on toy classification problem (large network).

B.7 MNIST and CIFAR10

In this section, we show additional numerical experiments on the MNIST and CIFAR10 datasets. The details of these problems are summarized in Table 2.

Table 2: Details for MNIST and CIFAR10 Problems.

Problem	**Structure**	d
`MNIST`	$784 - C_{5,3} - C_{5,5} - 10 - 10$	990
`CIFAR10`	$1024, 3 - C_{5,3} - C_{5,5} - 16 - 32 - 10$	2312

$C_{k,ch}$: convolution with kernel k and ch output channels.

B.7.1 Performance of Methods on MNIST

In this section, we present more experiments showing the accuracy and objective function value of different methods on the MNIST problem.

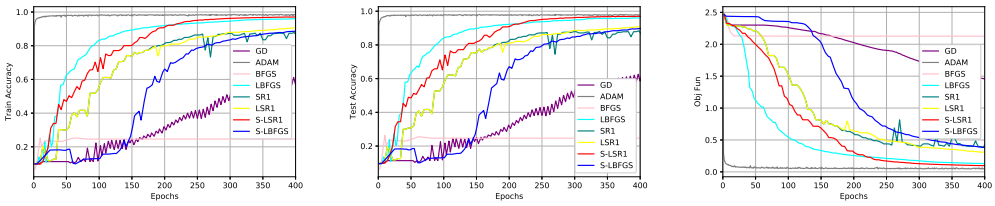

Figure 23: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on MNIST problems.

B.7.2 Performance of ADAM on MNIST

In this section, we show the performance of ADAM with different steplenghts on the MNIST problem. As is clear from the results in Figure 24, the performance of well-tuned ADAM is very good, however, when the steplength is not chosen correctly, the performance of ADAM can be terrible. Note, we have omitted runs for which ADAM diverged (i.e., when the steplength was chosen to be too large).

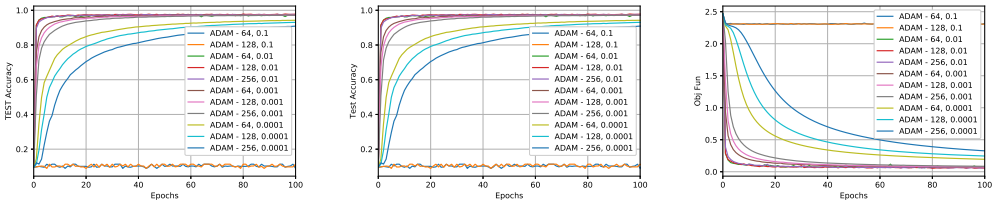

Figure 24: Performance of ADAM with different steplengths on MNIST.

B.7.3 Performance of Methods on CIFAR10

In this section, we present more experiments showing the accuracy and objective function value of different methods on the CIFAR10 problem.

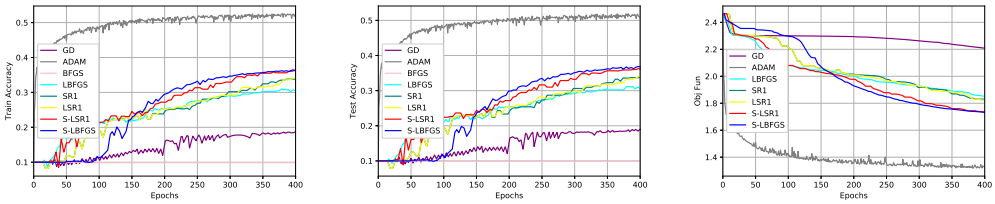

Figure 25: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on CIFAR problems.

C Distributed Computing and Cost

C.1 Cost of Communication

In this section, we show experiments conducted on a HPC cluster using a Cray Aries High Speed Network. The bandwidth ranges depending on the distance between nodes. We compiled the C++ code with the provided cray compiler.

In Figure 26, we show how the duration (seconds) of Broadcast and Reduce increases when vectors of longer length are processed.

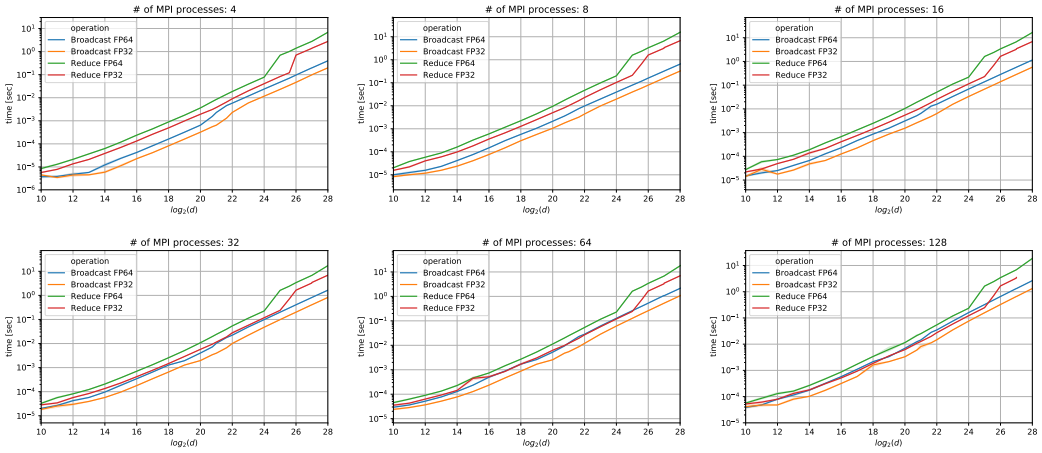

Figure 26: Duration of Broadcast and Reduce for various number of MPI processes and different length of the vector.

In Figure 27, we show how long it takes (seconds) to perform Broadcast and Reduce operations for vectors of a given length if performed on different numbers of MPI processes. We have performed each operation 100 times and are showing the average time and 95% confidence intervals.

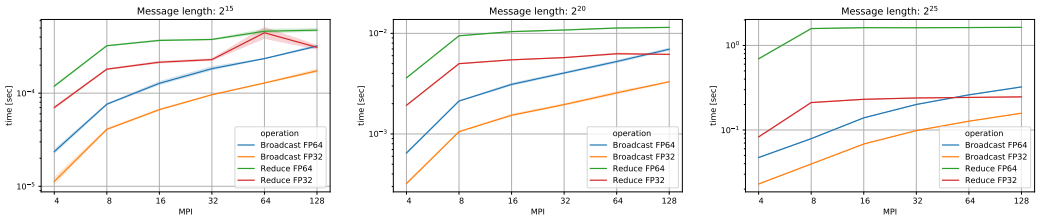

Figure 27: Duration of Broadcast and Reduce as a function of # of MPI processes for various lengths of vectors.

C.2 Distributed Computing Experiment Details

Table 3 summarizes the networks used in Section 4 and Figure 3.

Table 3: Deep Neural Networks used in the experiments.[6]

Model	d	Input	# classes
LeNet	3.2M	$28 \times 28 \times 3$	10
alexnet v2	50.3M	$224 \times 224 \times 3$	1,000
vgg a	132.8M	$224 \times 224 \times 3$	1,000

C.3 Cost, Storage and Parallelization

The cost per iteration of the different quasi-Newton methods can be deconstructed as follows: (1) the cost of computing the gradient, and (2) the cost of forming the search direction and taking the step. Note, motivated by the results in Figure 3, we assume that the cost computing a function value, gradient and Hessian vector product is comparable and is $\mathcal{O}(nd)$. The cost of computing the gradient is common for each method, whereas the search directions are computed differently for BFGS-type methods and SR1-type methods. More specifically, for BFGS methods we employ a line search and for SR1 method we use trust-region and solve the subproblem (2.4) using CG [45]. We denote the number of line search iterations and CG iterations as κ_{ls} and κ_{tr}, respectively. Table 4 summarizes the computational cost and storage for the different quasi-Newton methods.

Table 4: Summary of Computational Cost and Storage (per iteration) for different Quasi-Newton methods.

Method	Computational cost	Storage
BFGS	$nd + d^2 + \kappa_{ls} nd$	d^2
LBFGS	$nd + 4md + \kappa_{ls} nd$	$2md$
S-LBFGS	$nd + mnd + 4md + \kappa_{ls} nd$	-
SR1	$nd + d^2 + nd + \kappa_{tr} d^2$	d^2
LSR1	$nd + nd + \kappa_{tr} md$	$2md$
S-LSR1	$nd + mnd + nd + \kappa_{tr} md$	-

As is clear from Table 4, the proposed sampled quasi-Newton methods do not have a significantly higher cost per iteration than the classical limited memory variants of the methods. In the regime where $m \ll n, d$, the computational cost of the methods are $\mathcal{O}(nd)$. Moreover, the sampled quasi-Newton methods do not have any storage requirements. We should also note, that several computations that are required in our proposed methods are easily parallelizeable. These computations are the gradient evaluations, the function evaluations and the construction of the gradient displacement curvature pairs y.

[6] The structure of the deep neural network is taken from: https://github.com/tensorflow/models/tree/master/research/slim.