Sampled Quasi-Newton Methods for Deep Learning

Albert S. Berahas ALBERTBERAHAS@GMAIL.COM
Majid Jahani MAJ316@LEHIGH.EDU
Martin Takáč TAKAC.MT@GMAIL.COM
Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015

Abstract

We present two sampled quasi-Newton methods: sampled LBFGS and sampled LSR1. Contrary to the classical variants that sequentially build Hessian approximations, 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. We provide convergence guarantees for our proposed methods, and illustrate their performance in practice.

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).

In the last decades, much effort has been devoted to the development of stochastic first-order methods [6, 19, 20, 27, 30, 40, 45, 47] 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. 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 [3, 7, 31, 46, 51]. In order to alleviate these issues, stochastic Newton [5, 12, 25, 37, 46, 52] and stochastic quasi-Newton [2, 13, 16, 24, 28, 39, 48] 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 [3, 28, 46, 51].

This paper focuses on (full) batch methods that incorporate local second-order (curvature) information of the objective function. Specifically, we propose two variants of classical quasi-Newton methods[1] that sample a small number of random points at every iteration to build Hessian approximations.

We are motivated by the results presented in Figure 1 that illustrate the performance (for 10 different starting points) of several first- and second-order methods on a toy neural network classification task, given budget; see Section 4 for details. As is clear from the results, first-order

1. For a literature review of deterministic and stochastic quasi-Newton methods see Appendix A

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.

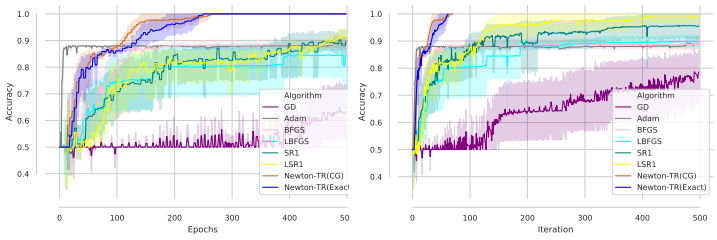

Figure 1: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, Newton-TR(CG, Exact) on a toy problem.

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.

2. 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*.

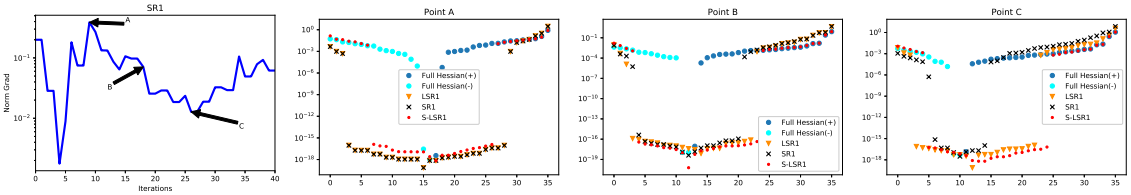

Figure 2: Comparison of the eigenvalues of (L)SR1 and S-LSR1 (@ A, B, C) for a toy classification problem.

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 C.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 2.1 and D.2.

2.1. 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. 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 Hessian vector products. Forming the y curvature pairs in this way has two main benefits: (1) it 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, and (2) is scale invariant.

Algorithm 1 Compute new (S, Y) curvature pairs

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

for $i = 1, 2, ..., m$ **do**
  Sample a random direction σ_i
  Construct $\bar{w} = w + r\sigma_i$
  Set $s = w - \bar{w}$ and $y = \nabla^2 F(w)s$
  Set $S = [S\ s]$ and $Y = [Y\ y]$
end for

Output: S, Y

2.2. Sampled LBFGS (S-LBFGS)

At the kth iteration, the S-LBFGS method (Algorithm 2) computes a new iterate via

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

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 [42] that is updated at every iteration using the set of curvature pairs sampled by Algorithm 1.

Algorithm 2 Sampled LBFGS (S-LBFGS)

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

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

Algorithm 2 is almost identical to the classical (L)BFGS algorithm [42]; 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. The latter, possibly the important feature of the method, 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.

2.3. Sampled LSR1 (S-LSR1)

At the kth iteration, the S-LSR1 method computes a new iterate via $w_{k+1} = w_k + p_k$, where p_k is the minimizer of the following subproblem

$$\min_{\|p\|\leq\Delta_k} m_k(p) = F(w_k) + \nabla F(w_k)^T p + \tfrac{1}{2} p^T B_k p, \qquad (2.2)$$

Δ_k is the trust region radius and B_k is the SR1 Hessian approximation [42] that is updated at every iteration by using the set of curvature pairs sampled by Algorithm 1.

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 C.3 for the details.

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 (2.2)
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** $w_{k+1} = w_k + p_k$
6: **else** $w_{k+1} = w_k$
7: $\Delta_{k+1} = \texttt{adjustTR}(\Delta_k, \rho_k)$ [see Appendix C.3]
8: **end for**

3. Convergence Analysis

In this section, we present convergence analyses for the sampled quasi-Newton methods for nonconvex functions. For brevity, we omit the proofs from the paper; see Appendix B for the proofs. See [4] for convergence results for strongly convex functions.

Sampled LBFGS For nonconvex functions, it is known that the (L)BFGS method can fail [17, 38]. To establish convergence in the nonconvex setting several techniques have been proposed [32, 33, 44]. 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

$$s^T y \geq \epsilon \|s\|^2, \tag{3.1}$$

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 standard assumptions:

Assumption 3.1 *The function F is twice continuously differentiable, and is bounded below by a scalar $\widehat{F}$. Moreover, the gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Lemma 3.2 *Suppose that Assumption 3.1 holds. 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 (3.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (3.1). 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 3.3).

Theorem 3.3 *Suppose that Assumption 3.1 holds. 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 (3.1), for some $\epsilon > 0$, where $0 < \alpha_k = \alpha \leq \frac{\mu_1}{\mu_2^2 L}$, and w_0 is the starting point. Then, for any $\tau > 1$, $\frac{1}{\tau}\sum_{k=0}^{\tau-1} \|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha \mu_1 \tau} \xrightarrow{\tau \to \infty} 0$.*

Sampled LSR1 In order to establish convergence results one needs to ensure that the SR1 Hessian update equation 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

$$|s^T(y - Bs)| \geq \epsilon \|s\| \|y - Bs\|, \tag{3.2}$$

where $\epsilon > 0$ is a predetermined constant. For the analysis in this section, we make the following assumption that implies that at every iteration the subproblem is solved sufficiently accurately.

Assumption 3.4 *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 $\beta_k = 1 + \|B_k\|$.*

Lemma 3.5 *Suppose that Assumption 3.1 holds. 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 (3.2), for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (3.2). 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 3.6).

Theorem 3.6 *Suppose that Assumption 3.1 holds. 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 (3.2), for some $\epsilon > 0$. Then, $\lim_{k\to\infty} \|\nabla F(w_k)\| = 0$.*

4. Numerical Experiments

In this section, we present numerical experiments on a toy classification problem. For implementation details and experiments on popular benchmarking neural network training tasks, see Appendix C.4 and C.6, respectively. Moreover, for details about the computational cost see Appendix D.

A Toy Classification Problem Consider the simple classification problem, illustrated in Figure 3, 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 C.5, Table 1 for details. For this problem, we ran each method 100 times starting from different initial points and show the results for different budget levels in Figure 4. As is clear from the figures, the proposed methods outperform their classical variants as well as the first-order methods. See Appendix C.5 and [4] for more results.

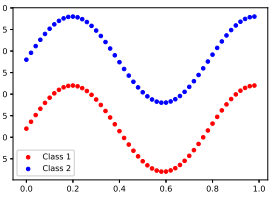

Figure 3: Toy Problem

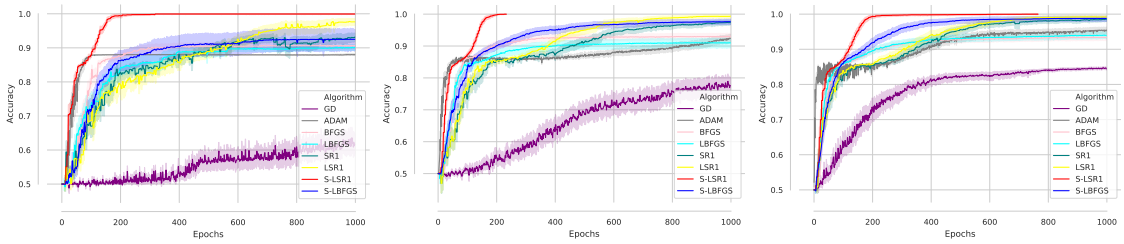

Figure 4: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problems. Networks: `small` (left); `medium` (right); `large` (right).

5. 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.

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 and Martin Takáč. A robust multi-batch l-bfgs method for machine learning. *Optimization Methods and Software*, pages 1–29, 2019.

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

[4] Albert S Berahas, Majid Jahani, and Martin Takáč. Quasi-newton methods for deep learning: Forget the past, just sample. *arXiv preprint arXiv:1901.09997*, 2019.

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

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

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

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

[9] 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.

[10] 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.

[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, 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.

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

[24] 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.

[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] Majid Jahani, Mohammadreza Nazari, Sergey Rusakov, Albert S Berahas, and Martin Takáč. Scaling up quasi-newton algorithms: Communication efficient distributed sr1. *arXiv preprint arXiv:1905.13096*, 2019.

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

[28] 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.

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

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

[33] 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.

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

[35] 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.

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

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

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

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

[40] 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.

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

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

[43] 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.

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

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

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

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

[48] 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.

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

[50] 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.

[51] 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.

[52] 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.

Appendix A. Brief Literature Review

Quasi-Newton methods, such as BFGS [8, 21, 23, 49] and SR1 [11, 14, 29] and their limited-memory variants LBFGS [34, 41] and LSR1 [9, 36], 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., [1, 13, 48]. 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 [13, 24, 35]. Specifically, we use Hessian vector products in lieu of gradient displacements.

Possibly the closest works to ours are Block BFGS [22] and its stochastic variant [24]. 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.

Appendix B. Theoretical Results and Proofs

We first restate the Assumptions that we use in the Convergence Analysis section (Section 3). We them prove all the results that appear in the main paper (Lemmas 3.2 & 3.5; Theorems 3.3 & 3.6).

B.1. Assumptions

Assumption 3.1 *The function F is twice continuously differentiable. The function $F(w)$ is bounded below by a scalar $\widehat{F}$. The gradients of F are L-Lipschitz continuous for all $w \in \mathbb{R}^d$.*

Assumption 3.4 *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 $\beta_k = 1 + \|B_k\|$.

B.2. Proof of Lemma 3.2

Lemma 3.2 *Suppose that Assumption 3.1 holds. 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 (3.1), for some $\epsilon > 0$, and $H_k = I$ if no curvature pairs satisfy (3.1). Then, there exist constants $0 < \mu_1 \leq \mu_2$ such that*

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

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 (B.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 (3.1) 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, there are two options for updating the curvature pairs $s_{k,j}$ and $y_{k,j}$:

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

for $j = 1, \ldots, m$. Let $\tilde{m}_k \in \{1, ..., m\}$ denote the number of curvature pairs that satisfy (3.1) at the kth iteration, where m is the memory.

In this setting, the skipping mechanism (3.1) 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 (3.1). We have from (3.1) 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

$$s_{k,j}^T y_{k,j} \leq \|s_{k,j}\| \|y_{k,j}\| \leq \frac{1}{\epsilon} \|y_{k,j}\|^2 \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{s_{k,j}^T y_{k,j}} \geq \epsilon. \tag{B.3}$$

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

$$\begin{aligned} y_{k,j}^T s_{k,j} &\geq \epsilon \|s_{k,j}\|^2 \\ &\geq \epsilon \frac{\|y_{k,j}\|^2}{L} \quad \Rightarrow \quad \frac{\|y_{k,j}\|^2}{s_{k,j}^T y_{k,j}} \leq \frac{L^2}{\epsilon}. \end{aligned} \tag{B.4}$$

Combining (B.3) and (B.4), we have

$$\epsilon \leq \frac{\|y_{k,j}\|^2}{y_{k,j}^T s_{k,j}} \leq \frac{L^2}{\epsilon}. \tag{B.5}$$

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,

$$\begin{aligned} Tr(B_{k+1}) &= Tr(B_k^{(0)}) - Tr\sum_{i=1}^{\tilde{m}_k}\left(\frac{B_k^{(i-1)}s_{k,i}s_{k,i}^T B_k^{(i-1)}}{s_{k,i}^T B_k^{(i-1)}s_{k,i}}\right) + Tr\sum_{i=1}^{\tilde{m}_k}\frac{y_{k,i}y_{k,i}^T}{y_{k,i}^T s_{k,i}} \\ &\leq Tr(B_k^{(0)}) + \sum_{i=1}^{\tilde{m}_k}\frac{\|y_{k,i}\|^2}{y_{k,i}^T s_{k,i}} \\ &\leq Tr(B_k^{(0)}) + \tilde{m}_k\frac{L^2}{\epsilon} \\ &\leq Tr(B_k^{(0)}) + m\frac{L^2}{\epsilon} \leq C_1, \end{aligned} \tag{B.6}$$

for some positive constant C_1, where the inequalities above are due to (B.5), 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 [43], the determinant of the matrix B_{k+1} generated by the sampled LBFGS method can be expressed as,

$$\begin{aligned} \det(B_{k+1}) &= \det(B_k^{(0)})\prod_{i=1}^{\tilde{m}_k}\frac{y_{k,i}^T s_{k,i}}{s_{k,i}^T B_k^{(i-1)}s_{k,i}} \\ &= \det(B_k^{(0)})\prod_{i=1}^{\tilde{m}_k}\frac{y_{k,i}^T s_{k,i}}{s_{k,i}^T s_{k,i}}\frac{s_{k,i}^T s_{k,i}}{s_{k,i}^T B_k^{(i-1)}s_{k,i}} \\ &\geq \det(B_k^{(0)})\left(\frac{\epsilon}{C_1}\right)^{\tilde{m}_k} \\ &\geq \det(B_k^{(0)})\left(\frac{\epsilon}{C_1}\right)^{m} \geq C_2, \end{aligned} \tag{B.7}$$

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 (B.5).

The trace (B.6) and determinant (B.7) 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.

■

B.3. Proof of Theorem 3.3

Theorem 3.3 *Suppose that Assumption 3.1 holds. 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 (3.1)*, for some* $\epsilon > 0$*, and* $H_k = I$ *if no curvature pairs satisfy* (3.1)*, where*

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

and w_0 *is the starting point. Then,*

$$\lim_{k\to\infty} \|\nabla F(w_k)\| \to 0, \tag{B.8}$$

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 have that

$$\begin{aligned}
F(w_{k+1}) &= F(w_k - \alpha H_k \nabla F(w_k)) \\
&\leq F(w_k) + \nabla F(w_k)^T(-\alpha H_k \nabla F(w_k)) + \frac{L}{2}\|\alpha H_k \nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F(w_k) + \frac{\alpha^2\mu_2^2 L}{2}\|\nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\mu_1\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2 L}{2}\|\nabla F(w_k)\|^2 \\
&= F(w_k) - \alpha\left(\mu_1 - \alpha\frac{\mu_2^2 L}{2}\right)\|\nabla F(w_k)\|^2 \\
&\leq F(w_k) - \alpha\frac{\mu_1}{2}\|\nabla F(w_k)\|^2,
\end{aligned} \tag{B.9}$$

where the first inequality is due to Assumption 3.1, the second and third inequalities arise as a consequence of Lemma 3.2 and the last inequality is due to the choice of the steplength.

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 3.1). Using the above, we have

$$\sum_{k=0}^{\tau-1}\|\nabla F(w_k)\|^2 \leq \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1}. \tag{B.10}$$

Taking limits we obtain,

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

which implies (B.8). Dividing (B.10) by τ we conclude

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

■

B.4. Proof of Lemma 3.5

Lemma 3.5 *Suppose that Assumptions 3.1 and 3.4 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 (3.2), for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (3.2). Then, there exists a constant $\nu_2 > 0$ such that*

$$\|B_k\| \leq \nu_2, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{B.11}$$

Proof As in the proof of Lemma 3.2, 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 (B.11) 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 (3.2) 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

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla F(w_k) - \nabla F(\bar{w}_j) \qquad \text{Option I,} \tag{B.12}$$

$$s_{k,j} = w_k - \bar{w}_j, \qquad y_{k,j} = \nabla^2 F(w_k)s_k \qquad \text{Option II.} \tag{B.13}$$

Given a set of $\tilde{m}_k$ curvature pairs that satisfy (3.2), 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

$$\|B_k^{(i)}\| \leq \left(1+\frac{1}{\epsilon}\right)^i \gamma_k + \left[\left(1+\frac{1}{\epsilon}\right)^i - 1\right]\bar{\gamma}_k. \tag{B.14}$$

For $i = 0$, the bound holds trivially since $B_k^{(0)} = \gamma_k I$. Now assume that (B.14) holds true for some $i \geq 0$. Note that all the curvature pairs that are used in the update of the Hessian approximation satisfy (3.2). 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 (3.2), 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 (B.14),

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

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.

■

B.5. Proof of Theorem 3.6

Theorem 3.6 *Suppose that Assumptions 3.1 and 3.4 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 3.2, for some $\epsilon > 0$, and $B_k = I$ if no curvature pairs satisfy (3.2). 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

$$\|\nabla F(w_{t_i})\| \geq 2\delta > 0 \tag{B.15}$$

for some $\epsilon > 0$ and for all i. Theorem 6.4.5 from [15] 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

$$\|\nabla F(w_k)\| \geq \delta, \quad \text{for} \quad t_i \leq k < \ell_i \quad \text{and} \quad \|\nabla F(w_{\ell_i})\| < \delta. \tag{B.16}$$

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 3.4, the fact that $\mathcal{K} \subseteq \mathcal{S}$ and (B.16), we deduce that for $k \in \mathcal{K}$

$$F(w_k) - F(w_k) \geq \eta_1[m_k(0) - m_k(p_k)] \geq \xi\delta\eta_1 \min\left[\frac{\delta}{\nu_2 + 1}, \Delta_k\right] \tag{B.17}$$

where we used the result of Lemma 3.5. Since the sequence $\{F(w_k)\}$ is monotonically decreasing and bounded below (Assumption 3.1), it is convergent, and the left-hand-side of (B.17) must tend to zero as $k \to \infty$. Thus,

$$\lim_{k\to\infty,\, k\in\mathcal{K}} \Delta_k = 0. \tag{B.18}$$

As a consequence, the term containing Δ_k is the dominant term in the $\min$ (B.17) and we have, for $k \in \mathcal{K}$ sufficiently large,

$$\Delta_k \leq \frac{F(w_k) - F(w_{k+1})}{(\nu_2 + 1)\delta\eta_1}. \tag{B.19}$$

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

$$\|w_{t_i} - w_{\ell_i}\| \leq \sum_{j=t_i,\, j\in\mathcal{K}}^{\ell_i - 1} \|w_j - w_{j+1}\| \leq \sum_{j=t_i,\, j\in\mathcal{K}}^{\ell_i - 1} \Delta_j \leq \frac{F(w_{t_i}) - F(w_{\ell_i})}{(\nu_2 + 1)\delta\eta_1}. \tag{B.20}$$

As a consequence of Assumption 3.1 and the monotonicity of the sequence $\{F(w_k)\}$, we have that the right-hand-side of (B.20) must converge to zero, and thus $\|w_{t_i} - w_{\ell_i}\| \to 0$ as $i \to \infty$.

By continuity of the gradient (Assumption 3.1), 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 (B.15) can exist, and the theorem is proved. ■

Appendix C. Additional Numerical Experiments and Method Details

In this section, we present additional numerical results and expand on some details about the methods.[2]

C.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.

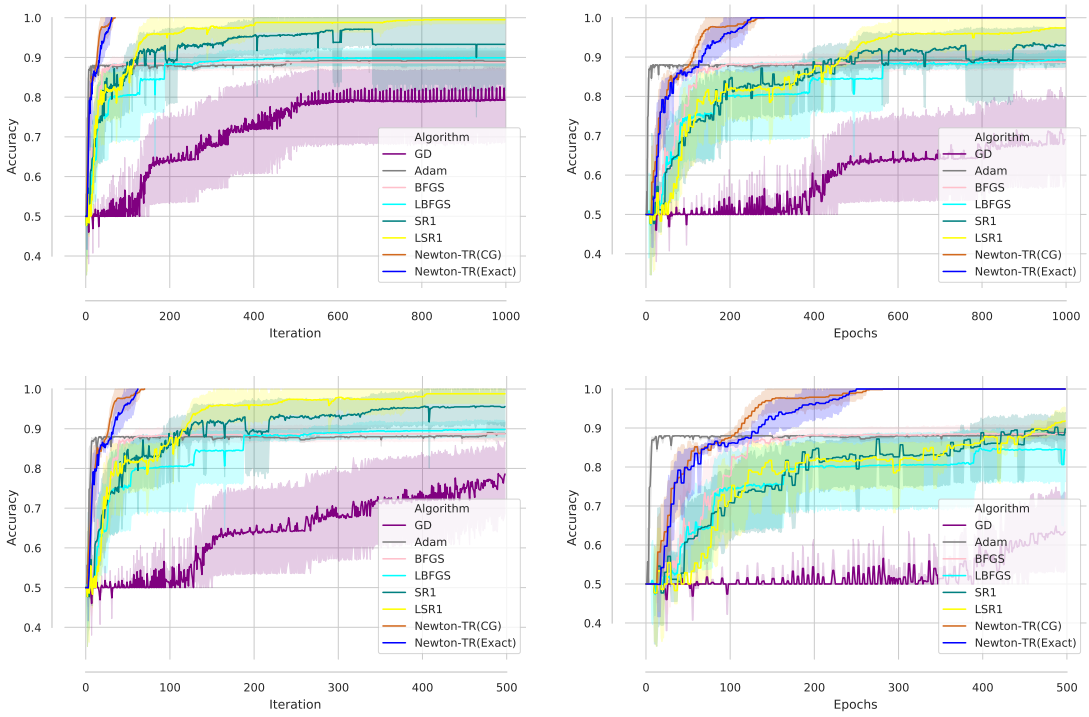

Figure 5: Comparison of Gradient Descent (GD), ADAM, BFGS, LBFGS, SR1, LSR1, Newton-TR (Exact), Newton-TR (CG) on a toy classification problem in terms of iterations and epochs.

2. 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.

C.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.

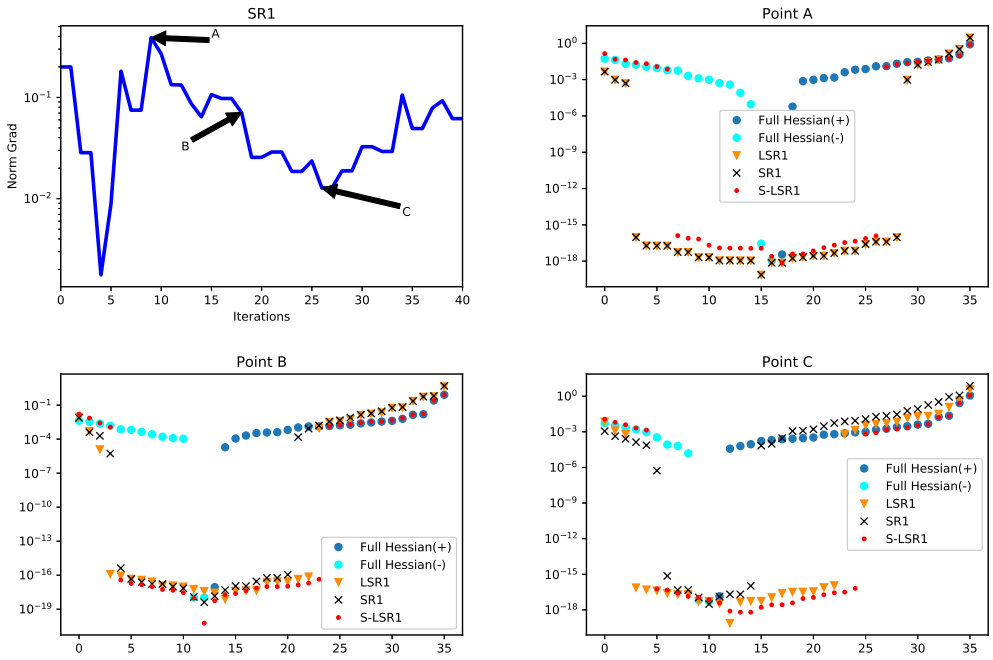

Figure 6: Comparison of the eigenvalues of SR1, LSR1 and S-LSR1 at different points for a toy classification problem.

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.

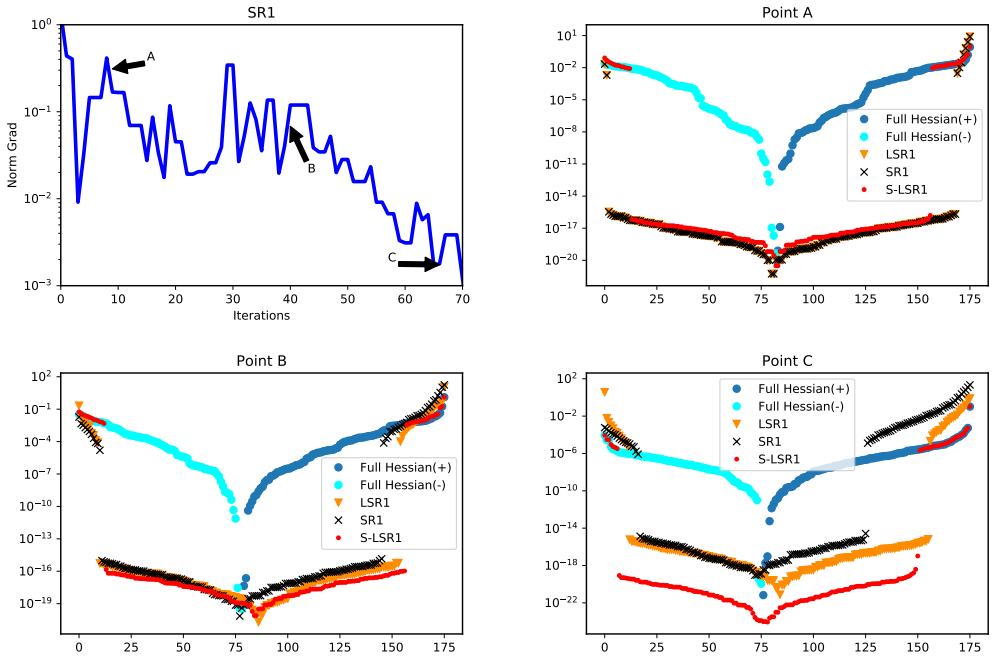

Figure 7: Comparison of the eigenvalues of SR1, LSR1 and S-LSR1 at different points for a toy classification problem.

Note: for Figure 6 we used $T = 40$, $m = 16$ and $r = 0.01$, and for Figure 7 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.

C.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 [42] 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

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

In this section, we discuss the implementation details for all the methods.[3]

- 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 [42].
- For **SR1**-type methods, we solved the trust-region subproblems (2.2) using CG-Steihaug [42].
- 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 [42].
- Implementing the **limited memory SR1** methods is not trivial; we made use of the compact representations of the SR1 matrices [10] and computed the steps dynamically.

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.

C.5. Toy Example

In this section, we present additional numerical results for the toy classification problem described in Section 4. In the following experiments, we ran each method from 100 different initial points. The details of the three networks are summarized in Table 1.

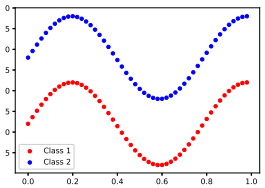

Figure 8: Toy Classification Problem

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

C.5.1. 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.

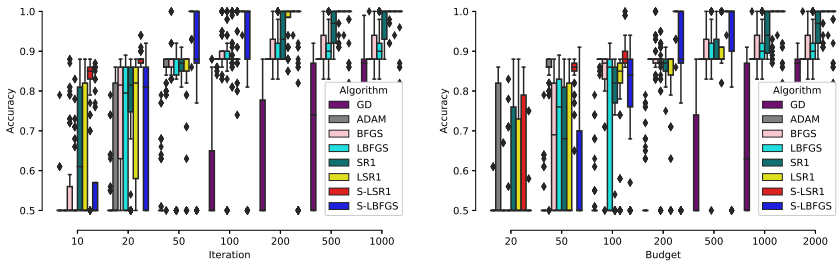

Figure 9: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (small network).

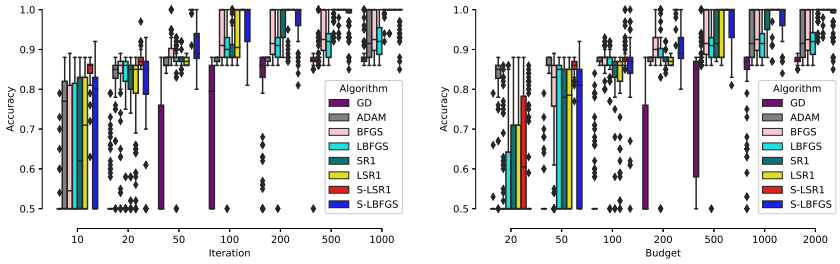

Figure 10: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (medium network).

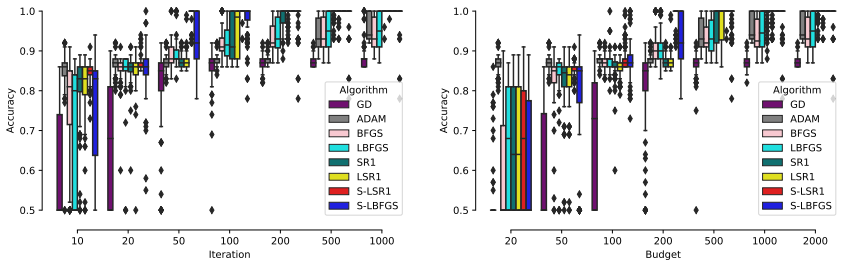

Figure 11: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (large network).

C.5.2. 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.

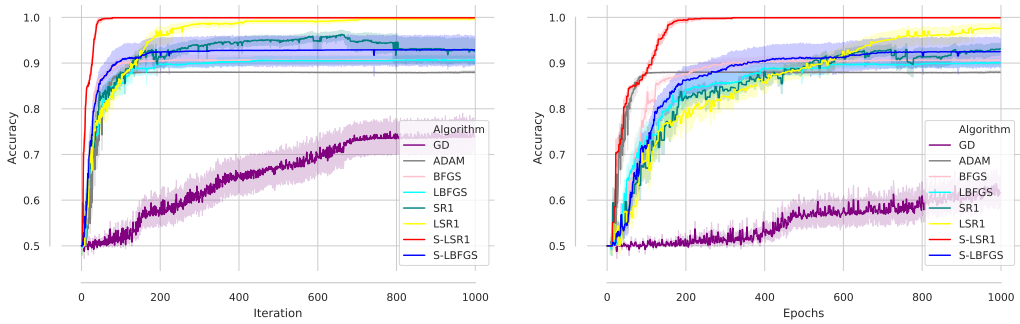

Figure 12: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (small network).

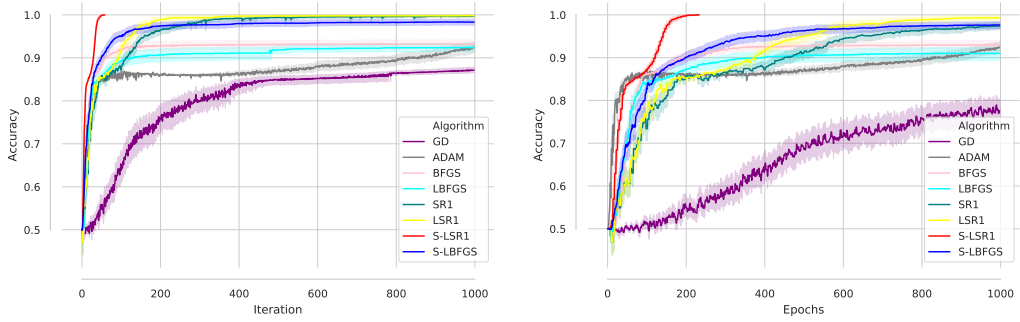

Figure 13: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (medium network).

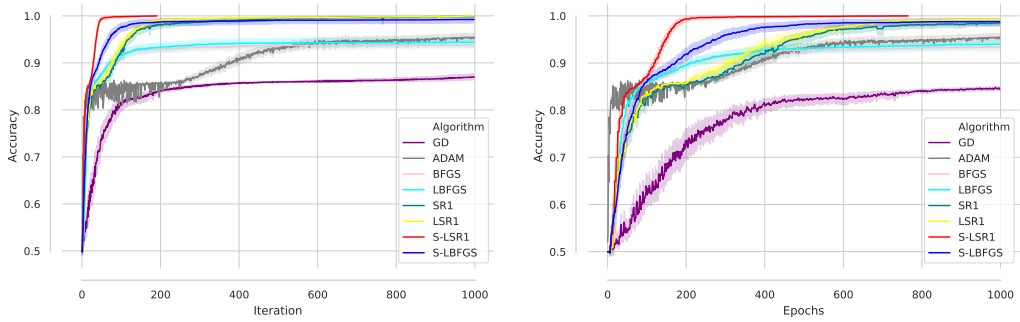

Figure 14: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problem (large network).

C.5.3. 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.

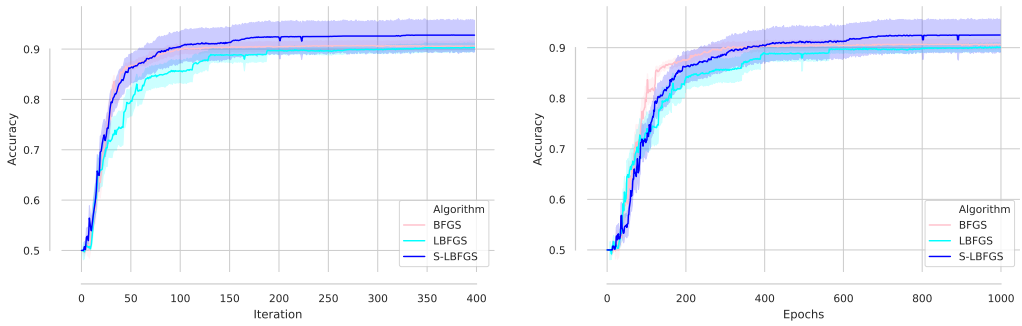

Figure 15: Performance of BFGS-type methods on toy classification problem (small network).

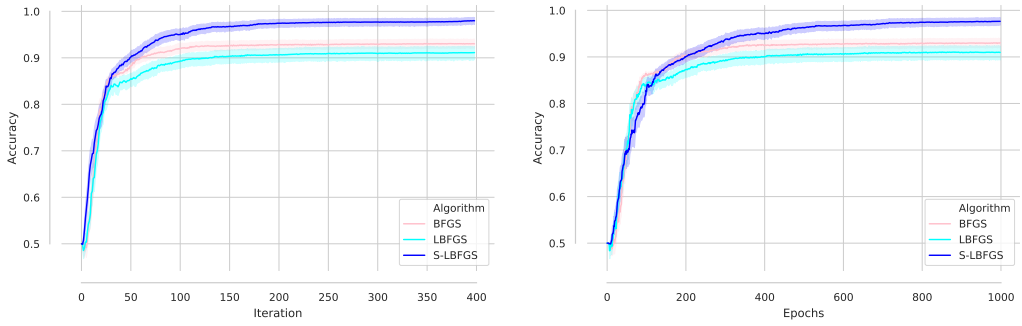

Figure 16: Performance of BFGS-type methods on toy classification problem (medium network).

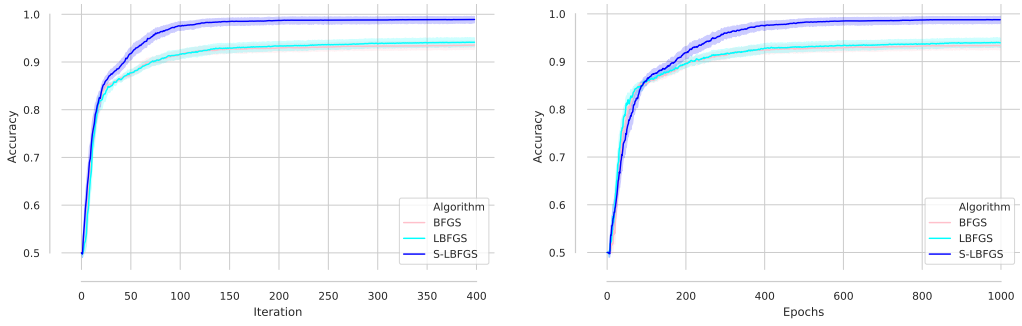

Figure 17: Performance of BFGS-type methods on toy classification problem (large network).

C.5.4. 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.

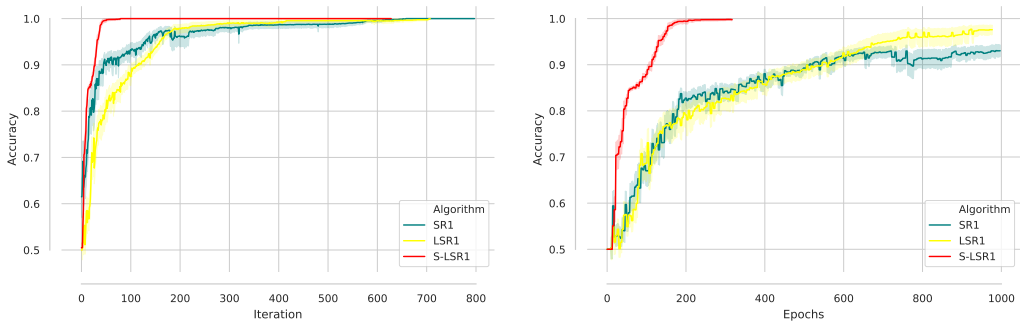

Figure 18: Performance of SR1-type methods on toy classification problem (small network).

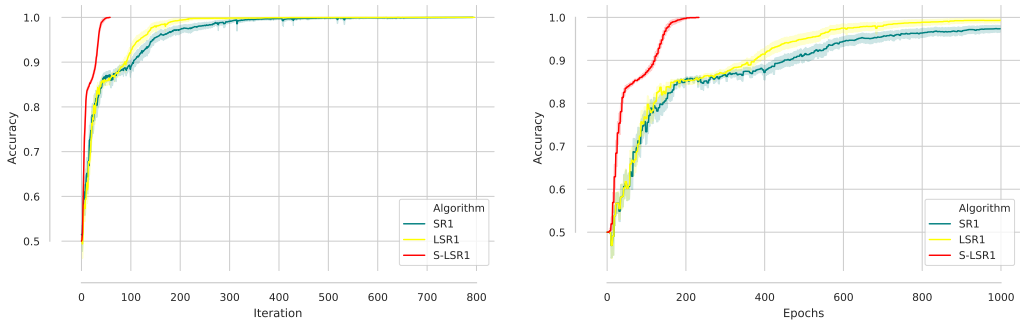

Figure 19: Performance of SR1-type methods on toy classification problem (medium network).

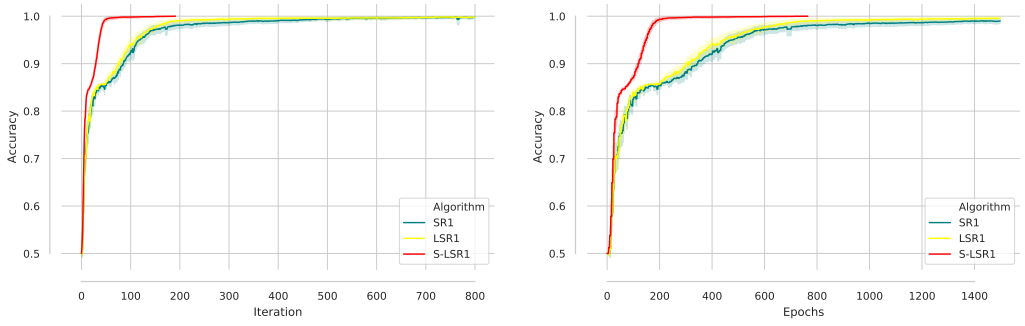

Figure 20: Performance of SR1-type methods on toy classification problem (large network).

C.6. 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.

C.6.1. 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.

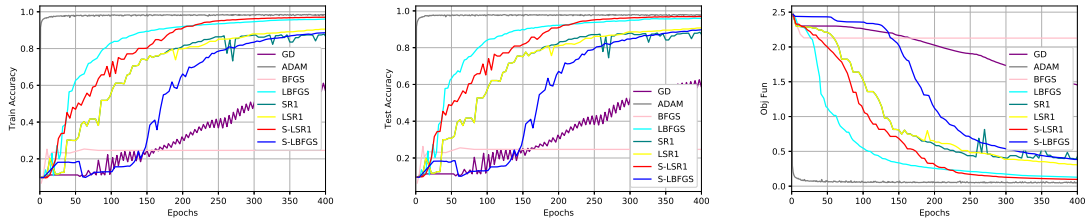

Figure 21: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on MNIST problems.

C.6.2. 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 22, 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).

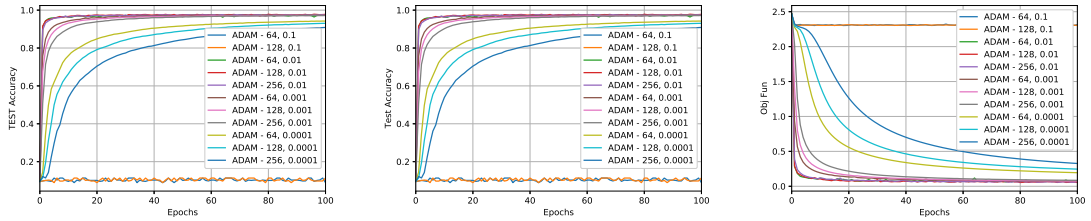

Figure 22: Performance of ADAM with different steplengths on MNIST.

C.6.3. 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.

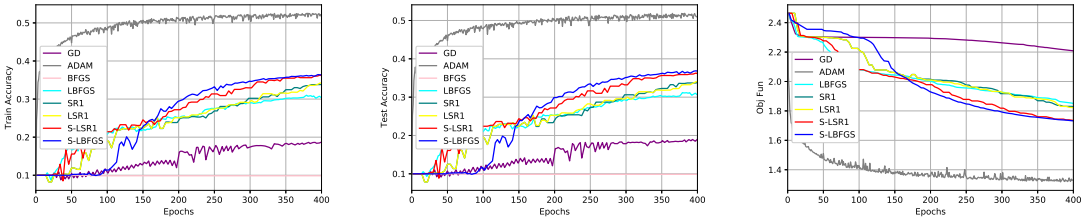

Figure 23: Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on CIFAR problems.

Appendix D. Distributed Computing and Cost

In this section, we first discuss why the cost of communication can be an issue, and then describe the communication and computation cost of our proposed methods. For a fully distributed implementation of the sampled LSR1 method see [26].

D.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 24, we show how the duration (seconds) of Broadcast and Reduce increases when vectors of longer length are processed.

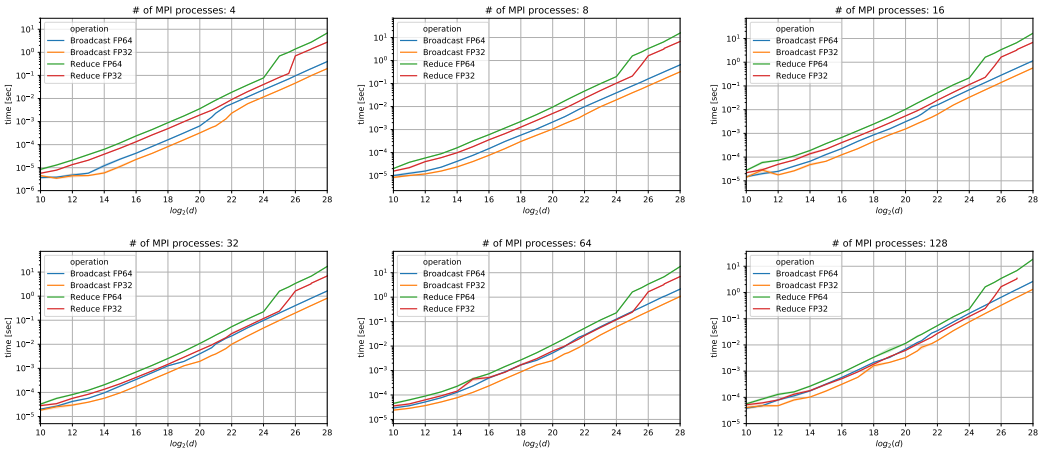

Figure 24: Duration of Broadcast and Reduce for various number of MPI processes and different length of the vector.

In Figure 25, 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.

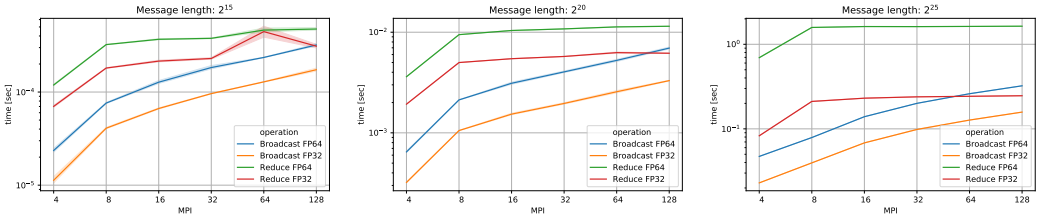

Figure 25: Duration of Broadcast and Reduce as a function of # of MPI processes for various lengths of vectors.

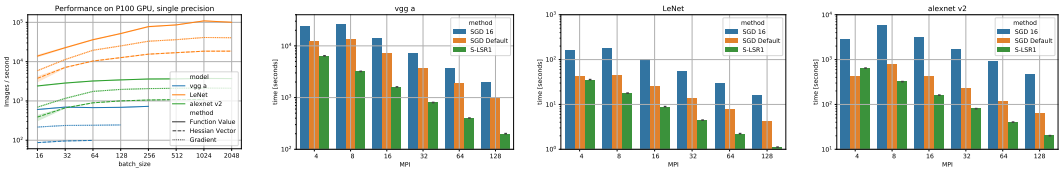

Figure 26: Performance (Images/second) as a function of batch size for different DNN models and operations on a single NVIDIA Tesla P100 GPU (left). Time (sec) to complete 1 epoch of SG and to perform 1 iteration of S-LSR1 on a dataset with 1M images using varying number of MPI processes (bar charts).

D.2. 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 26 (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 Table 3. 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 [18, 50]. Moreover, we observe that for these networks the cost of computing function values, gradients and Hessian vector products is comparable[4]. In Figure 26 (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[5]: (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 D.1 for more details.

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

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 26, 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 (2.2) using CG [42]. We denote the number of line search iterations and CG iterations

4. We assume that the cost of computing function values, gradients and Hessian vector products is $\mathcal{O}(nd)$.
5. 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

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.