A Robust Multi-Batch L-BFGS Method for Machine Learning

Albert S. Berahas[a*] and Martin Takáč[a]

[a]*Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA*

(*v5.0 released July 2015*)

This paper describes an implementation of the L-BFGS method designed to deal with two adversarial situations. The first occurs in distributed computing environments where some of the computational nodes devoted to the evaluation of the function and gradient are unable to return results on time. A similar challenge occurs in a multi-batch approach in which the data points used to compute function and gradients are purposely changed at each iteration to accelerate the learning process. Difficulties arise because L-BFGS employs gradient differences to update the Hessian approximations, and when these gradients are computed using different data points the updating process can be unstable. This paper shows how to perform stable quasi-Newton updating in the multi-batch setting, studies the convergence properties for both convex and nonconvex functions, and illustrates the behavior of the algorithm in a distributed computing platform on binary classification logistic regression and neural network training problems that arise in machine learning.

Keywords: L-BFGS, multi-batch, fault-tolerant, sampling, consistency, overlap.

AMS Subject Classification: 90C30; 90C06; 90C53.

1. Introduction

It is common in machine learning to encounter optimization problems involving tens of millions of training examples and millions of variables. To deal with the demands of time, storage and processing power imposed by such applications, high performance implementations of stochastic gradient and batch quasi-Newton methods have been developed; see e.g., [2, 4, 22, 23, 58, 66, 69]. In this paper we study a batch approach based on the L-BFGS method [43, 53] that strives to reach the right balance between efficient learning and productive parallelism.

At present, due to its fast learning properties and low per-iteration cost, the preferred method for very large scale applications is the stochastic gradient (SG) method [13, 60], and its variance-reduced and accelerated variants [24, 32, 33, 36, 37, 42, 51, 52, 61]. These methods are implemented either in an asynchronous manner (e.g., using a parameter server in a distributed setting) or following a synchronous mini-batch approach that exploits parallelism in the gradient evaluations [7, 29, 38, 58, 59, 65]. A drawback of the asynchronous approach is that it cannot use large batches, as this would cause updates to become too dense and compromise the stability and scalability of the method [44, 58]. As a result, the algorithm spends more time in communication as compared to computation.

This work substantially extends [5] published at the Neural Information Processing Systems (NeurIPS) conference in 2016.

*Corresponding author. Email: albertberahas@u.northwestern.edu

On the other hand, using a synchronous mini-batch approach one can achieve a near-linear decrease in the number of SG iterations as the mini-batch size is increased, up to a certain point after which the increase in computation is not offset by the faster convergence [65].

An alternative to SG-type methods are batch methods, such as L-BFGS [53], because they parallelize well and are able to achieve high training accuracy. Batch methods allow for more computation per node, so as to achieve a better balance with the communication costs [4, 68]; however, batch methods are not as efficient learning algorithms as SG methods in a sequential setting [14, 31]. To benefit from both types of methods, some high performance machine learning systems implement both types of methods [2, 23], and algorithms that transition from the stochastic to the batch regime [9, 10, 17, 26] have also received attention recently.

The goal of this paper is to propose a single method that selects a *sizeable* subset (batch) of the training data to compute a step and changes this batch at every iteration to improve the learning abilities of the method. In order to differentiate it from the mini-batch approach used in conjunction with the SG method, which employs a very small subset of the training data, we call this the *multi-batch* approach. In this regime it is natural to employ a quasi-Newton method, as incorporating second-order information imposes little computational overhead and improves the stability and speed of the method. However, the multi-batch approach can cause difficulties to quasi-Newton methods as these methods employ gradient differences to update the Hessian approximations.

More specifically, in this paper we study how to design a *robust multi-batch* implementation of the limited-memory version of the classical BFGS method [15, 25, 28, 63]—which we call the *multi-batch L-BFGS* method—in the presence of two adverse situations [5]. The first occurs in parallel implementations when some of the computational nodes devoted to the evaluation of the function and gradient are unable to return results on time, i.e., in the presence of *faults*. This amounts to using different data points to evaluate the function and gradient at the beginning and the end of the iteration, which can be harmful to quasi-Newton methods since they employ gradient differences to update Hessian approximations. A similar challenge occurs in a *multi-batch* approach in which the data points used to compute the function and gradient are purposely changed at each iteration (or every several iterations) to accelerate the learning process. The main objective of this paper is to show that *stable quasi-Newton updating* can be achieved in these settings without incurring extra computational cost or special synchronization. The key is to perform quasi-Newton updating based on the *overlap* between consecutive batches. The only restriction is that this overlap should not be insignificant, something that can be expected, or easily enforced, in most situations.

Recently, several stochastic quasi-Newton (SQN) methods have been proposed; see e.g., [5, 11, 18, 20, 30, 34, 47, 62, 67]. The methods enumerated above differ in three major aspects: (*i*) the update rules for the curvature (correction) pairs and the Hessian approximation, (*ii*) the frequency of updating, and (*iii*) the required extra computational cost and synchronization required. Our method is different from these methods predominantly due to the fact that it does not modify the BFGS update equations or the form of the curvature pairs, and does not require extra (gradient) computations. Additionally, our method is designed to work in a distributed settings with faults, in which faults occur randomly and sample consistency cannot be assumed, and as such several SQN methods are not suitable.

We analyze the convergence properties of the multi-batch L-BFGS method using a fixed step length strategy, as well as a diminishing step length strategy, on both strongly convex and nonconvex problems. This is appropriate in our setting, as using a fixed step

length approach is popular in practice, and facilitates the study of the stability of quasi-Newton updating in a distributed setting. For strongly convex functions, we show that the algorithm converges, at a linear rate, to an approximate solution whose accuracy depends on the variance of the gradients and the step length. In the nonconvex setting, we show that if cautious BFGS updating is employed, the expected value of the average norm-squared of the gradient is bounded.

We present numerical experiments on a plethora of problems that arise in machine learning and deep learning. We first illustrate the robustness of our proposed approach on binary classification logistic regression problems on a distributed computing platform with faults and in the serial multi-batch setting. The results indicate that the proposed method achieves a good balance between computation and communication costs. Moreover, we present results on neural network training tasks that illustrate that when larger batch-size is used, our algorithm is competitive with the state-of-the-art. Finally, we demonstrate the strong and weak scaling properties of the proposed method.

The paper is organized as follows. In Section 2 we describe the multi-batch L-BFGS method in detail. In Section 3 we provide convergence analyses for the proposed method for strongly convex and nonconvex functions. Numerical results that illustrate the practical performance and robustness of the multi-batch L-BFGS method are reported in Section 4. Finally, in Section 5 we provide some concluding remarks.

2. A Multi-Batch Quasi-Newton Method

Ideally, in supervised learning, one seeks to minimize expected risk, defined as

$$R(w) = \int_{\Omega} f(w; x, y) dP(x, y) = \mathbb{E}[f(w; x, y)], \tag{2.1}$$

where (x, y) are input-output pairs, $f : \mathbb{R}^d \to \mathbb{R}$ is the composition of a prediction function (parametrized by w) and a loss function, and Ω is the space of input-output pairs endowed with a probability distribution $P(x, y)$. Since the distribution P is typically not known, one approximates (2.1) by the empirical risk

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} f(w; x^i, y^i) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

where (x^i, y^i), for $i = 1, \ldots, n$, denote the training examples, also referred to as data points or samples. The training problem consists of finding an optimal choice of the parameters $w \in \mathbb{R}^d$ with respect to F, i.e., to compute a solution of the problem

$$\min_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{2.2}$$

In a pure batch approach, one applies a gradient-based method to the deterministic optimization problem (2.2). In this regime, a popular method is L-BFGS [43, 53]. When n is large, it is natural to parallelize the computation of F and ∇F by assigning the evaluation of component functions f_i, or subsets of the component functions, to different processors. If this is done on a distributed computing platform, it is possible for some of the computational nodes, dedicated to a portion of the evaluation of the objective

function and the gradient, to be slower than the rest. In this case, the contribution of the slow (or unresponsive) computational nodes could potentially be ignored given the stochastic nature of the true objective function (2.1). However, this leads to an inconsistency in the objective function and gradient at the beginning and at the end of the iteration, which can be detrimental to quasi-Newton methods, as mentioned above. Hence, we seek to develop a *fault-tolerant* version of the batch L-BFGS method that is capable of dealing with slow or unresponsive computational nodes.

A similar challenge arises in a *multi-batch* implementation of the L-BFGS method in which only a subset of the data is used to compute the gradient at every iteration. We consider a method in which the dataset is randomly divided into a number of batches and the minimization is performed with respect to a different batch at every iteration. Specifically, at the k-th iteration the algorithm chooses $S_k \subset \{1, \ldots, n\}$, computes

$$F^{S_k}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} f_i(w_k), \qquad g_k^{S_k} = \nabla F^{S_k}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(w_k), \tag{2.3}$$

and takes a step along the direction $-H_k g_k^{S_k}$, where H_k is an approximation to $\nabla^2 F(w_k)^{-1}$. Allowing the sample S_k to change freely at every iteration gives this approach flexibility and is beneficial to the learning process. Note, we refer to S_k as the sample of training points, even though S_k only indexes those points.

The case of unresponsive computational nodes and the multi-batch regime are similar in nature, i.e., the samples S_k used change from one iteration to the next. The main difference is that node failures create unpredictable changes to the samples, whereas a multi-batch method has control over the sample generation. In either case, the algorithm employs a stochastic approximation to the gradient and can no longer be considered deterministic. We must, however, distinguish our setting from that of the classical SG method, which employs small mini-batches. Our algorithm operates with much larger batches so that distributing the computation of the function and gradient is beneficial, and the compute time is not overwhelmed by communication costs. This gives rise to gradients with relatively small variance and justifies the use of a second-order method such as L-BFGS.

The robust implementation of the L-BFGS method, proposed in [5], is based on the following observation: The difficulties created by the use of a different sample S_k at each iteration can be circumvented if consecutive samples S_k and S_{k+1} have an overlap, so that

$$O_k = S_k \cap S_{k+1} \neq \emptyset.$$

One can then perform *stable quasi-Newton updating* by computing gradient differences based on this overlap, i.e., by defining

$$y_{k+1} = g_{k+1}^{O_k} - g_k^{O_k}, \qquad s_{k+1} = w_{k+1} - w_k, \tag{2.4}$$

in the notation given in (2.3), and using this correction pair (y_k, s_k) in the BFGS update. When the overlap set O_k is not too small, y_k is a useful approximation of the curvature of the objective function along the most recent displacement, and leads to a productive quasi-Newton step. This observation is based on an important property of Newton-like methods, namely that there is much more freedom in choosing a Hessian approximation than in computing the gradient [3, 8, 16, 45]. More specifically, a smaller sample O_k can

be employed for updating the inverse Hessian approximation H_k, than for computing the batch gradient $g_k^{S_k}$ used to define the search direction $-H_k g_k^{S_k}$. In summary, by ensuring that unresponsive nodes do not constitute the vast majority of all compute nodes in a fault-tolerant parallel implementation, or by exerting a small degree of control in the creation of the samples S_k in the multi-batch regime, one can design a robust method that naturally builds upon the fundamental properties of BFGS updating.

We should mention that a commonly used fix for ensuring stability of quasi-Newton updating in machine learning is to enforce gradient consistency [47, 62], i.e., to use the same sample S_k to compute gradient evaluations at the beginning and the end of the iteration, at the cost of double gradient evaluations. Another popular remedy is to use the same batch S_k for multiple iterations [50], alleviating the gradient inconsistency problem at the price of slower convergence. In this paper, we assume that such *sample consistency is not possible* (the fault-tolerant case) or *desirable* (the multi-batch regime), and wish to design and analyze an implementation of L-BFGS that imposes minimal restrictions in the changes of the sample.

2.1 *Specification of the Method*

Let us begin by considering a robust implementation of the multi-batch BFGS method and then consider its limited memory version. At the k-th iteration, the multi-batch BFGS algorithm chooses a set $S_k \subset \{1, \dots, n\}$ and computes a new iterate by the formula

$$w_{k+1} = w_k - \alpha_k H_k g_k^{S_k}, \tag{2.5}$$

where α_k is the step length, $g_k^{S_k}$ is the batch gradient (2.3) and H_k is the inverse BFGS Hessian matrix approximation that is updated at every iteration by means of the formula

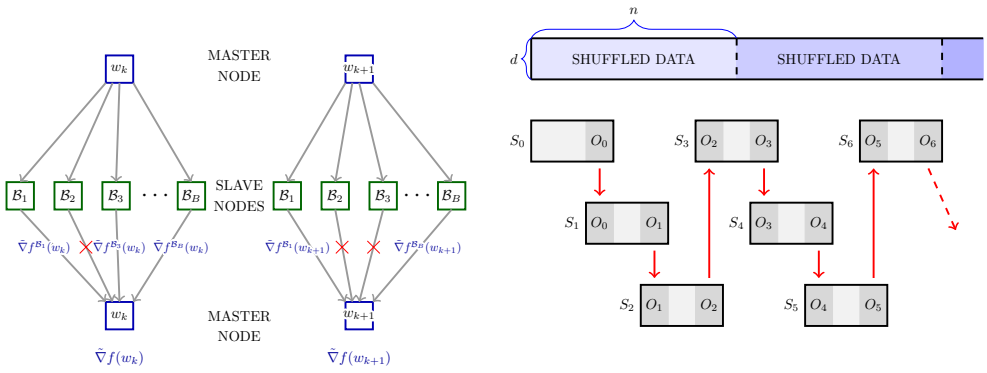
$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \qquad \rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = 1 - \rho_k y_k s_k^T.$$

To compute the correction vectors (s_k, y_k), we determine the overlap set $O_k = S_k \cap S_{k+1}$ consisting of the samples that are common at the k-th and $k+1$-st iterations. We define

$$g_k^{O_k} = \nabla F^{O_k}(w_k) = \frac{1}{|O_k|} \sum_{i \in O_k} \nabla f_i(w_k),$$

and compute the correction pairs as in (2.4). This completely specifies the algorithm, except for the choice of step length α_k; in this paper we consider constant and diminishing step lengths.

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 a set of m correction pairs $\{s_i, y_i\}$ kept in storage. The memory parameter m is typically in the range 2 to 20. When computing the search direction (matrix-vector product) in (2.5) it is not necessary to form the dense matrix H_k since one can obtain this product via the two-loop recursion [54], using the m most recent correction pairs. Employing this mechanism, the search direction can be computed in $\mathcal{O}(d)$ floating operations, where d is the number of variables. After the step has been computed, the oldest pair (s_j, y_j) is discarded and the new curvature pair is stored.



(a) Fault-tolerant sampling (b) Multi-batch sampling

Figure 1. Sample and Overlap formation for two adversarial situations.

A pseudo-code of the multi-batch limited-memory BFGS algorithm is given in Algorithm 1, and depends on several parameters. The parameter r denotes the fraction of samples in the dataset used to define the gradient, i.e., $r = \frac{|S|}{n}$. The parameter o denotes the length of overlap between consecutive samples, and is defined as a fraction of the number of samples in a given batch S, i.e., $o = \frac{|O|}{|S|}$.

Algorithm 1 Multi-Batch L-BFGS

Input: w_0 (initial iterate), m (memory parameter), r (batch, fraction of n), o (overlap, fraction of batch), $k \leftarrow 0$ (iteration counter).

1: Create initial batch S_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Calculate the search direction $p_k = -H_k g_k^{S_k}$
4: Choose a step length $\alpha_k > 0$
5: Compute $w_{k+1} = w_k + \alpha_k p_k$
6: Create the next batch S_{k+1}
7: Compute the curvature pairs $s_{k+1} = w_{k+1} - w_k$ and $y_{k+1} = g_{k+1}^{O_k} - g_k^{O_k}$
8: Replace the oldest pair (s_i, y_i) by s_{k+1}, y_{k+1} (if m pairs stored, else just add)
9: **end for**

2.2 *Sample Generation*

The *fault-tolerant* and *multi-batch* settings differ in the way the samples S_k and O_k are formed (Lines 1 & 6, Algorithm 1). In the former, sampling is done automatically as a by-product of the nodes that fail to return a computation (gradient evaluation). In the latter, the samples S_k and O_k used at every iteration are purposefully changed in order to accelerate the learning process, thus sampling is user controlled. In either setting, independent sampling can be achieved; a necessary condition to establish convergence results. We first describe the *fault-tolerant* setting, and then propose two sampling strategies that can be employed in the *multi-batch* setting. Let $T = \{(x^i, y^i), \text{ for } i = 1, \ldots, n\}$ denote the training set.

Fault-Tolerant Consider a distributed implementation in which slave nodes read the current iterate w_k from the master node, compute a local gradient on a subset of the dataset, and send it back to the master node for aggregation in the calculation (2.3). Given a time (computational) budget, it is possible for some nodes to fail to return a result. The schematic in Figure 1(a) illustrates the gradient calculation across two iterations, k and $k+1$, in the presence of faults. Here, B is the total number of slave nodes, $\mathcal{B}_i$ for $i = 1, ..., B$ denote the batches of data that each slave node i receives ($T = \cup_i \mathcal{B}_i$), and $\tilde{\nabla} f(w)$ is the gradient calculation using all nodes that responded within the preallocated time.

Let $\mathcal{J}_k \subset \{1, 2, ..., B\}$ and $\mathcal{J}_{k+1} \subset \{1, 2, ..., B\}$ be the set of indices of all nodes that returned a gradient at the k-th and $k+1$-st iterations, respectively. Using this notation $S_k = \cup_{j \in \mathcal{J}_k} \mathcal{B}_j$ and $S_{k+1} = \cup_{j \in \mathcal{J}_{k+1}} \mathcal{B}_j$, and we define $O_k = \cup_{j \in \mathcal{J}_k \cap \mathcal{J}_{k+1}} \mathcal{B}_j$. The simplest implementation in this setting preallocates the data on each compute node, requiring minimal data communication, i.e., only one data transfer. In this case, the samples S_k are independent if node failures occur randomly. On the other hand, if the same set of nodes fail, then the sample creation will be biased, which is harmful both in theory and in practice. One way to ensure independent sampling is to shuffle and redistribute the data to all nodes after every iteration or after a certain number of iterations.

Multi-Batch Sampling In the *multi-batch* setting several strategies can be employed, with the only restriction that consecutive batches S_k and S_{k+1} should, to a certain degree, overlap. We propose two sampling strategies: (i) overlaps O_k are forced in the sample creation process, (ii) the overlapping set O_k is subsampled from the batch S_k. In practice the two strategies perform on par, however, there is a subtle difference. In the second strategy the batches are sampled independently, something that is not true for the strategy in which overlapping samples are forced. The independent sampling strategy of course does not come for free as this strategy incurs an increase in computational cost per iteration. However, as mentioned above, the overlapping set O_k need not be very large, and thus the increase in cost is negligible as compared to the rest of the computation. We now describe the two approaches in more detail.

Figure 1(b) illustrates the sample creation process in the first strategy. The dataset is shuffled and batches are generated by collecting subsets of the training set, in order. Every set (except S_0) is of the form $S_k = \{O_{k-1}, N_k, O_k\}$, where O_{k-1} and O_k are the overlapping samples with batches S_{k-1} and S_{k+1} respectively, and N_k are the samples that are unique to batch S_k. After each pass through the dataset, the samples are reshuffled, and the procedure described above is repeated. In our implementation samples are drawn without replacement, guaranteeing that after every epoch (pass over the whole dataset) all samples are used. This strategy has the advantage that it requires no extra computation in the evaluation of $g_k^{O_k}$ and $g_{k+1}^{O_k}$, but the samples S_k are not independent.

The second sampling strategy is simpler and requires less control. At every iteration k, a batch S_k is created by randomly selecting $|S_k|$ elements from $\{1, \dots n\}$. The set O_k is then formed by randomly selecting $|O_k|$ elements from S_k (subsampling). Note, in this sampling strategy the samples O_k need not be in the set S_{k+1}. This strategy is slightly more expensive since $g_{k+1}^{O_k}$ requires extra computation, but if the overlap is small this cost is not significant.

3. Convergence Analysis

In this Section, we analyze the convergence properties of the multi-batch L-BFGS method (Algorithm 1) when applied to the minimization of *strongly convex* and *nonconvex* objective functions, using a fixed step length strategy, as well as a diminishing step length strategy. We assume that the goal is to minimize the empirical risk F (2.2), but note that a similar analysis could be used to study the minimization of the expected risk (2.1).

3.1 *Strongly Convex case*

Due to the stochastic nature of the multi-batch approach, every iteration of Algorithm 1 employs a gradient that contains errors that do not converge to zero. Therefore, by using a fixed step length strategy one cannot establish convergence to the optimal solution $w^\star$, but only convergence to a neighborhood of $w^\star$ [48]. Nevertheless, this result is of interest as it reflects the common practice of using a fixed step length and decreasing it only if the desired testing error has not been achieved. It also illustrates the tradeoffs that arise between the size of the batch and the step length.

In our analysis, we make the following assumptions about the objective function and the algorithm.

Assumptions A

(1) F is twice continuously differentiable.
(2) There exist positive constants $\hat{\lambda}$ and $\hat{\Lambda}$ such that $\hat{\lambda} I \preceq \nabla^2 F^O(w) \preceq \hat{\Lambda} I$ for all $w \in \mathbb{R}^d$ and all sets $O \subset \{1, 2, \ldots, n\}$ of length $|O| = o \cdot r \cdot n$.
(3) There exist constants $\gamma \geq 0$ and $\eta \geq 1$ such that $\mathbb{E}_S \left[\|\nabla F^S(w)\|^2 \right] \leq \gamma^2 + \eta \|\nabla F(w)\|^2$ for all $w \in \mathbb{R}^d$ and all sets $S \subset \{1, 2, \ldots, n\}$ of length $|S| = r \cdot n$.
(4) The samples S are drawn independently and $\nabla F^S(w)$ is an unbiased estimator of the true gradient $\nabla F(w)$ for all $w \in \mathbb{R}^d$, i.e., $\mathbb{E}_S[\nabla F^S(w)] = \nabla F(w)$.

Note that Assumption $A.2$ implies that the entire Hessian $\nabla^2 F(w)$ also satisfies

$$\lambda I \preceq \nabla^2 F(w) \preceq \Lambda I, \quad \forall w \in \mathbb{R}^d, \tag{3.1}$$

for some constants $\lambda, \Lambda > 0$. Assuming that every subsampled function $F^O(w)$ is strongly convex is not unreasonable as a regularization term is commonly added in practice when that is not the case.

We begin by showing that the inverse Hessian approximations H_k generated by the multi-batch L-BFGS method have eigenvalues that are uniformly bounded above and away from zero. The proof technique used is an adaptation of that in [18].

LEMMA 3.1 *If Assumptions A.1 & A.2 hold, there exist constants $0 < \mu_1 \leq \mu_2$ such that the inverse Hessian approximations $\{H_k\}$ generated by Algorithm 1 satisfy*

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

Proof. Instead of analyzing the inverse Hessian approximation H_k, we study the Hessian approximation $B_k = H_k^{-1}$. In this case, the limited memory quasi-Newton updating formula is given as follows:

(1) Set $B_k^{(0)} = \frac{y_k^T y_k}{s_k^T y_k} I$ and $\tilde{m} = \min\{k, m\}$; where m is the memory in L-BFGS.
(2) For $i = 0, ..., \tilde{m} - 1$ set $j = k - \tilde{m} + 1 + i$ and compute

$$B_k^{(i+1)} = B_k^{(i)} - \frac{B_k^{(i)} s_j s_j^T B_k^{(i)}}{s_j^T B_k^{(i)} s_j} + \frac{y_j y_j^T}{y_j^T s_j}.$$

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

The curvature pairs s_k and y_k are updated via the following formulae

$$y_{k+1} = g_{k+1}^{O_k} - g_k^{O_k}, \qquad s_{k+1} = w_{k+1} - w_k. \tag{3.2}$$

A consequence of Assumption $A.2$ is that the eigenvalues of any sub-sampled Hessian ($|O|$ samples) are bounded above and away from zero. Utilizing this fact, the convexity of component functions and the definitions (3.2), we have

$$y_k^T s_k \geq \frac{1}{\hat{\Lambda}} \|y_k\|^2 \quad \Rightarrow \quad \frac{\|y_k\|^2}{y_k^T s_k} \leq \hat{\Lambda}. \tag{3.3}$$

On the other hand, strong convexity of the sub-sampled functions, the consequence of Assumption $A.2$ and definitions (3.2), provide a lower bound,

$$y_k^T s_k \leq \frac{1}{\hat{\lambda}} \|y_k\|^2 \quad \Rightarrow \quad \frac{\|y_k\|^2}{y_k^T s_k} \geq \hat{\lambda}. \tag{3.4}$$

Combining the upper and lower bounds (3.3) and (3.4)

$$\hat{\lambda} \leq \frac{\|y_k\|^2}{y_k^T s_k} \leq \hat{\Lambda}. \tag{3.5}$$

The above proves that the eigenvalues of the matrices $B_k^{(0)} = \frac{y_k^T y_k}{s_k^T y_k} I$ at the start of the L-BFGS update cycles are bounded above and away from zero, for all k. 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, and set $j_i = k - \tilde{m} + i$. The trace of the matrix B_{k+1} can be expressed as,

$$\begin{aligned} tr(B_{k+1}) &= tr\left(B_k^{(0)}\right) - tr \sum_{i=1}^{\tilde{m}} \left(\frac{B_k^{(i-1)} s_{j_i} s_{j_i}^T B_k^{(i-1)}}{s_{j_i}^T B_k^{(i-1)} s_{j_i}} \right) + tr \sum_{i=1}^{\tilde{m}} \frac{y_{j_i} y_{j_i}^T}{y_{j_i}^T s_{j_i}} \\ &\leq tr\left(B_k^{(0)}\right) + \sum_{i=1}^{\tilde{m}} \frac{\|y_{j_i}\|^2}{y_{j_i}^T s_{j_i}} \leq tr\left(B_k^{(0)}\right) + \tilde{m}\hat{\Lambda} \leq C_1, \end{aligned} \tag{3.6}$$

for some positive constant C_1, where the inequalities above are due to (3.5), and the fact that the eigenvalues of the initial L-BFGS matrix $B_k^{(0)}$ are bounded above and away from zero.

Using a result due to [56], the determinant of the matrix B_{k+1} generated by the multi-batch L-BFGS method can be expressed as,

$$\det(B_{k+1}) = \det\left(B_k^{(0)}\right)\prod_{i=1}^{\tilde{m}}\frac{y_{j_i}^T s_{j_i}}{s_{j_i}^T B_k^{(i-1)} s_{j_i}} = \det\left(B_k^{(0)}\right)\prod_{i=1}^{\tilde{m}}\frac{y_{j_i}^T s_{j_i}}{s_{j_i}^T s_{j_i}}\frac{s_{j_i}^T s_{j_i}}{s_{j_i}^T B_k^{(i-1)} s_{j_i}}$$
$$\geq \det\left(B_k^{(0)}\right)\left(\frac{\hat{\lambda}}{C_1}\right)^{\tilde{m}} \geq C_2, \tag{3.7}$$

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 Assumption $A.2$.

The trace (3.6) and determinant (3.7) inequalities derived above imply that the largest eigenvalues of all matrices B_k are bounded from above, uniformly, and that the smallest eigenvalues of all matrices B_k are bounded away from zero, uniformly. ■

Before we present the main theorem for the multi-batch L-BFGS method that employs constant step lengths, we state one more intermediate Lemma that bounds the distance between the function value at any point $w \in \mathbb{R}^d$ and the optimal function value with respect to the norm of the gradient squared.

LEMMA 3.2 *Let Assumptions A.1 & A.2 hold, and let $F^\star = F(w^\star)$, where $w^\star$ is the minimizer of F. Then, for all $w \in \mathbb{R}^d$,*

$$2\lambda(F(w) - F^\star) \leq \|\nabla F(w)\|^2.$$

Proof. As a result of Assumptions $A.1$, $A.2$ and (3.1), for all $x, y \in \mathbb{R}^d$

$$F(x) \leq F(y) + \nabla F(y)^T(x-y) + \frac{1}{2\lambda}\|\nabla F(y) - \nabla F(x)\|^2;$$

see [49, Chapter 2.1.3]. Let $x = w$ and $y = w^\star$

$$F(w) \leq F^\star + \nabla F(w^\star)^T(w - w^\star) + \frac{1}{2\lambda}\|\nabla F(w) - \nabla F(w^\star)\|^2$$
$$\leq F^\star + \frac{1}{2\lambda}\|\nabla F(w)\|^2.$$

Re-arranging the above expression yields the desired result. ■

Utilizing Lemmas 3.1 and 3.2, we show that the multi-batch L-BFGS method with a constant step length converges linearly to a neighborhood of the optimal solution.

THEOREM 3.3 *Suppose that Assumptions A.1-A.4 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 1, where $\alpha_k = \alpha$ satisfies*

$$0 < \alpha \leq \frac{\mu_1}{\mu_2^2 \eta \lambda}, \tag{3.8}$$

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

$$\mathbb{E}\big[F(w_k) - F^\star\big] \leq \big(1 - \alpha\mu_1\lambda\big)^k \left[F(w_0) - F^\star\right] + \Big[1 - (1 - \alpha\mu_1\lambda)^k\Big] \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}$$
$$\xrightarrow{k\to\infty} \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}.$$

Proof. We have that

$$\begin{aligned} F(w_{k+1}) &= F(w_k - \alpha H_k \nabla F^{S_k}(w_k)) \\ &\leq F(w_k) + \nabla F(w_k)^T(-\alpha H_k \nabla F^{S_k}(w_k)) + \frac{\Lambda}{2}\|\alpha H_k \nabla F^{S_k}(w_k)\|^2 \\ &\leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F^{S_k}(w_k) + \frac{\alpha^2\mu_2^2\Lambda}{2}\|\nabla F^{S_k}(w_k)\|^2, \end{aligned} \tag{3.9}$$

where the first inequality arises due to (3.1), and the second inequality arises as a consequence of Lemma 3.1. Taking the expectation (over S_k) of equation (3.9)

$$\begin{aligned} \mathbb{E}_{S_k}[F(w_{k+1})] &\leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F(w_k) + \frac{\alpha^2\mu_2^2\Lambda}{2}\mathbb{E}_{S_k}\left[\|\nabla F^{S_k}(w_k)\|^2\right] \\ &\leq F(w_k) - \alpha\mu_1\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\Lambda}{2}\left(\gamma^2 + \eta\|\nabla F(w)\|^2\right) \\ &= F(w_k) - \alpha\left(\mu_1 - \frac{\alpha\mu_2^2\eta\Lambda}{2}\right)\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2} \qquad (3.10) \\ &\leq F(w_k) - \frac{\alpha\mu_1}{2}\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2} \qquad (3.11) \end{aligned}$$

where the first inequality makes use of Assumption $A.4$, the second inequality arises due to Lemma 3.1 and Assumption $A.3$ and the third inequality is due to the step length (3.8). Since F is λ-strongly convex, we can substitute the result of Lemma 3.2 in (3.11),

$$\begin{aligned} \mathbb{E}_{S_k}[F(w_{k+1})] &\leq F(w_k) - \frac{\alpha\mu_1}{2}\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2} \\ &\leq F(w_k) - \alpha\mu_1\lambda[F(w_k) - F^\star] + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2}. \end{aligned} \tag{3.12}$$

Let

$$\phi_k = \mathbb{E}[F(w_k) - F^\star], \tag{3.13}$$

where the expectation is over all batches $S_0, S_1, ..., S_{k-1}$ and all history starting with w_0. Equation (3.12) can be expressed as,

$$\phi_{k+1} \leq (1 - \alpha\mu_1\lambda)\phi_k + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2}. \tag{3.14}$$

Since the step length is chosen according to (3.8) we deduce that $0 \leq (1 - \alpha\mu_1\lambda) < 1$.

Subtracting $\frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}$ from either side of (3.14) yields

$$\begin{aligned}\phi_{k+1} - \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda} &\leq (1-\alpha\mu_1\lambda)\phi_k + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2} - \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda} \\ &= (1-\alpha\mu_1\lambda)\left[\phi_k - \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}\right]. \end{aligned} \tag{3.15}$$

Recursive application of (3.15) yields

$$\phi_k - \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda} \leq (1-\alpha\mu_1\lambda)^k\left[\phi_0 - \frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}\right],$$

and thus,

$$\phi_k \leq (1-\alpha\mu_1\lambda)^k\phi_0 + \left[1-(1-\alpha\mu_1\lambda)^k\right]\frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}.$$

Finally using the definition of ϕ_k (3.13) with the above expression yields the desired result

$$\mathbb{E}\left[F(w_k) - F^\star\right] \leq (1-\alpha\mu_1\lambda)^k\left[F(w_0) - F^\star\right] + \left[1-(1-\alpha\mu_1\lambda)^k\right]\frac{\alpha\mu_2^2\gamma^2\Lambda}{2\mu_1\lambda}.$$

■

The bound provided by this theorem has two components: (i) a term decaying linearly to zero, and (ii) a term identifying the neighborhood of convergence. Note, a larger step length yields a more favorable constant in the linearly decaying term, at the cost of an increase in the size of the neighborhood of convergence. We consider these tradeoffs further in Section 4, where we also note that larger batch sizes increase the opportunities for parallelism and improve the limiting accuracy in the solution, but slow down the learning abilities of the algorithm. We should also mention that unlike the first-order variant of the algorithm ($H_k = I$), the step length range prescribed by the multi-batch L-BFGS method depends on μ_1 and μ_2, the smallest and largest eigenvalues of the L-BFGS Hessian approximation. In the worst-case, the presence of the matrix H_k can make the limit in Theorem 3.3 significantly worse than that of the first-order variant if the update has been unfortunate and generates ill-conditioned matrices. We should note, however, such worst-case behavior is almost never observed in practice for BFGS updating.

One can establish convergence of the multi-batch L-BFGS method to the optimal solution $w^\star$ by employing a sequence of step lengths $\{\alpha_k\}$ that converge to zero according to the schedule proposed by [60]. However, that provides only a sub-linear rate of convergence, which is of little interest in our context where large batches are employed and some type of linear convergence is expected. In this light, Theorem 3.3 is more relevant to practice; nonetheless, we state the theorem here for completeness, and, for brevity, refer the reader to [18, Theorem 3.2] for more details and the proof.

THEOREM 3.4 *Suppose that Assumptions A.1-A.4 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 1 with*

$$\alpha_k = \frac{\beta}{k+1} \qquad \text{and} \qquad \beta > \frac{\mu_1}{\mu_2^2 \eta \lambda},$$

starting from w_0. *Then for all* $k \geq 0$,

$$\mathbb{E}\left[F(w_k) - F^\star\right] \leq \frac{Q(\beta)}{k+1},$$

where $Q(\beta) = \max\left\{\frac{\mu_2^2 \beta^2 \gamma^2 \Lambda}{2(2\mu_1 \lambda \beta - 1)}, F(w_0) - F^\star\right\}$.

Theorem 3.4 shows that, for strongly convex functions, the multi-batch L-BFGS method with an appropriate schedule of diminishing step lengths converges to the optimal solution at a sub-linear rate. We should mention that another way to establish convergence to the optimal solution for the multi-batch L-BFGS method is to employ variance reduced gradients [24, 33, 37, 42, 51, 52, 61]. In this setting, one can establish linear convergence to the optimal solution using constant step lengths. We defer the analysis of the multi-batch L-BFGS method that employs variance reduced gradients to a different study [6].

3.2 *Nonconvex case*

The BFGS method is known to fail on nonconvex problems [21, 46]. Even for L-BFGS, which makes only a finite number of updates at each iteration, one cannot guarantee that the Hessian approximations have eigenvalues that are uniformly bounded above and away from zero. To establish convergence of the (L-)BFGS method in the nonconvex setting several techniques have been proposed including *cautious* updating [41], *modified* updating [40] and *damping* [57]. Here we employ a cautious strategy that is well suited to our particular algorithm; we skip the Hessian update, i.e., set $H_{k+1} = H_k$, if the curvature condition

$$y_k^T s_k \geq \epsilon \|s_k\|^2 \tag{3.16}$$

is not satisfied, where $\epsilon > 0$ is a predetermined constant. On the other hand, sufficient curvature is guaranteed when the updates are not skipped. Using said mechanism, we show that the eigenvalues of the Hessian matrix approximations generated by the multi-batch L-BFGS method are bounded above and away from zero (Lemma 3.5). The analysis presented in this section is based on the following assumptions.

Assumptions B

(1) F *is twice continuously differentiable.*

(2) *The gradients of* F *are* Λ*-Lipschitz continuous for all* $w \in \mathbb{R}^d$*; the gradients of* F^S *are* Λ_S*-Lipschitz continuous for all* $w \in \mathbb{R}^d$ *and all sets* $S \subset \{1, 2, \ldots, n\}$ *of length* $|S| = r \cdot n$*; and, the gradients of* F^O *are* Λ_O*-Lipschitz continuous for all* $w \in \mathbb{R}^d$ *and all sets* $O \subset \{1, 2, \ldots, n\}$ *of length* $|O| = o \cdot r \cdot n$.

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

(4) There exist constants $\gamma \geq 0$ *and* $\eta \geq 1$ *such that* $\mathbb{E}_S\left[\|\nabla F^S(w)\|^2\right] \leq \gamma^2 + \eta\|\nabla F(w)\|^2$ *for all* $w \in \mathbb{R}^d$ *and all sets* $S \subset \{1, 2, \ldots, n\}$ *of length* $|S| = r \cdot n$.

(5) The samples S *are drawn independently and* $\nabla F^S(w)$ *is an unbiased estimator of the true gradient* $\nabla F(w)$ *for all* $w \in \mathbb{R}^d$, *i.e.,* $\mathbb{E}_S[\nabla F^S(w)] = \nabla F(w)$.

Similar to the strongly convex case, we first show that the eigenvalues of the L-BFGS Hessian approximations are bounded above and away from zero.

LEMMA 3.5 *Suppose that Assumptions B.1 & B.2 hold. Let* $\{H_k\}$ *be the inverse Hessian approximations generated by Algorithm 1, with the modification that the inverse Hessian approximation update is performed only when (3.16) is satisfied, for some* $\epsilon > 0$, *else* $H_{k+1} = H_k$. *Then, there exist constants* $0 < \mu_1 \leq \mu_2$ *such that*

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

Proof. Similar to the proof of Lemma 3.1, we study the direct Hessian approximation $B_k = H_k^{-1}$. The curvature pairs s_k and y_k are updated via the following formulae

$$y_{k+1} = g_{k+1}^{O_k} - g_k^{O_k}, \qquad s_{k+1} = w_{k+1} - w_k.$$

The skipping mechanism (3.16) provides both an upper and lower bound on the quantity $\frac{\|y_k\|^2}{y_k^T s_k}$, which in turn ensures that the initial L-BFGS Hessian approximation is bounded above and away from zero. The lower bound is attained by repeated application of Cauchy's inequality to condition (3.16). We have from (3.16) that

$$\epsilon\|s_k\|^2 \leq y_k^T s_k \leq \|y_k\|\|s_k\| \quad \Rightarrow \quad \|s_k\| \leq \frac{1}{\epsilon}\|y_k\|,$$

from which it follows that

$$s_k^T y_k \leq \|s_k\|\|y_k\| \leq \frac{1}{\epsilon}\|y_k\|^2 \quad \Rightarrow \quad \frac{\|y_k\|^2}{s_k^T y_k} \geq \epsilon. \tag{3.17}$$

The upper bound is attained by the Lipschitz continuity of sample gradients (Assumption *B.2*),

$$y_k^T s_k \geq \epsilon\|s_k\|^2 \geq \epsilon\frac{\|y_k\|^2}{\Lambda_O^2} \quad \Rightarrow \quad \frac{\|y_k\|^2}{s_k^T y_k} \leq \frac{\Lambda_O^2}{\epsilon}. \tag{3.18}$$

Combining (3.17) and (3.18),

$$\epsilon \leq \frac{\|y_k\|^2}{y_k^T s_k} \leq \frac{\Lambda_O^2}{\epsilon}.$$

The above proves that the eigenvalues of the matrices $B_k^{(0)} = \frac{y_k^T y_k}{s_k^T y_k} I$ at the start of the L-BFGS update cycles are bounded above and away from zero, for all k. The rest of the proof follows the same Trace-Determinant argument as in the proof of Lemma 3.1, the only difference being that the last inequality in (3.7) comes as a result of the cautious update strategy. ■

We now follow the analysis in [12, Chapter 4] to establish the following result about the behavior of the gradient norm for the multi-batch L-BFGS method with a cautious update strategy.

THEOREM 3.6 *Suppose that Assumptions B.1-B.5 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 1, with the modification that the inverse Hessian approximation update is performed only when (3.16) is satisfied, for some $\epsilon > 0$, else $H_{k+1} = H_k$, where $\alpha_k = \alpha$ satisfies*

$$0 < \alpha \leq \frac{\mu_1}{\mu_2^2 \eta \lambda},$$

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

$$\mathbb{E}\left[\frac{1}{\tau}\sum_{k=0}^{\tau-1}\|\nabla F(w_k)\|^2\right] \leq \frac{\alpha\mu_2^2\gamma^2\Lambda}{\mu_1} + \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1\tau} \xrightarrow{\tau\to\infty} \frac{\alpha\mu_2^2\gamma^2\Lambda}{\mu_1}.$$

Proof. Starting with (3.11) and taking the expectation over all batches $S_0, S_1, ..., S_{k-1}$ and all history starting with w_0 yields

$$\phi_{k+1} - \phi_k \leq -\frac{\alpha\mu_1}{2}\mathbb{E}\|\nabla F(w_k)\|^2 + \frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2}, \tag{3.19}$$

where $\phi_k = \mathbb{E}[F(w_k)]$. Summing (3.19) over the first τ iterations

$$\begin{aligned}\sum_{k=0}^{\tau-1}[\phi_{k+1} - \phi_k] &\leq -\frac{\alpha\mu_1}{2}\sum_{k=0}^{\tau-1}\mathbb{E}\|\nabla F(w_k)\|^2 + \sum_{k=0}^{\tau-1}\frac{\alpha^2\mu_2^2\gamma^2\Lambda}{2} \\ &= -\frac{\alpha\mu_1}{2}\mathbb{E}\left[\sum_{k=0}^{\tau-1}\|\nabla F(w_k)\|^2\right] + \frac{\alpha^2\mu_2^2\gamma^2\Lambda\tau}{2}. \end{aligned} \tag{3.20}$$

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

$$\sum_{k=0}^{\tau-1}[\phi_{k+1} - \phi_k] = \phi_\tau - \phi_0 = \mathbb{E}[F(w_\tau)] - F(w_0) \geq \widehat{F} - F(w_0).$$

Substituting the above expression into (3.20) and rearranging terms

$$\mathbb{E}\left[\sum_{k=0}^{\tau-1}\|\nabla F(w_k)\|^2\right] \leq \frac{\alpha\mu_2^2\gamma^2\Lambda\tau}{\mu_1} + \frac{2[F(w_0) - \widehat{F}]}{\alpha\mu_1}.$$

Dividing the above equation by τ completes the proof. ∎

This result bounds the average norm of the gradient of F after the first $\tau - 1$ iterations, and shows that, in expectation, the iterates spend increasingly more time in regions where the objective function has a small gradient. Under appropriate conditions, we can

establish a convergence rate for the multi-batch L-BFGS method with cautious updates to a stationary point of F, similar to the results proven for the SG method [27]. For completeness we state and prove the result.

THEOREM 3.7 *Suppose that Assumptions B.1-B.5 hold. Let $\{w_k\}$ be the iterates generated by Algorithm 1, with the modification that the inverse Hessian approximation update is performed only when (3.16) is satisfied, for some $\epsilon > 0$, else $H_{k+1} = H_k$. Let*

$$\alpha_k = \alpha = \frac{c}{\sqrt{\tau}}, \quad c = \sqrt{\frac{2(F(w_0) - \hat{F})}{\mu_2^2 \gamma^2 \Lambda}}, \quad \delta(\alpha) = \mu_1 - \frac{\alpha \mu_2^2 \eta \Lambda}{2},$$

where $\tau > \frac{c^2 \mu_2^4 \eta^2 \Lambda^2}{4\mu_1^2}$, and w_0 is the starting point. Then,

$$\min_{0 \le k \le \tau - 1} \mathbb{E}\left[\|\nabla F(w_k)\|^2\right] \le \sqrt{\frac{2(F(w_0) - \hat{F})\mu_2^2 \gamma^2 \Lambda}{\delta(\alpha)^2 \tau}}.$$

Proof. Starting with (3.10), we have

$$\begin{aligned} \mathbb{E}_{S_k}[F(w_{k+1})] &\le F(w_k) - \alpha \left(\mu_1 - \frac{\alpha \mu_2^2 \eta \Lambda}{2}\right) \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2} \\ &= F(w_k) - \alpha \delta(\alpha) \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2}, \end{aligned} \tag{3.21}$$

where $\delta(\alpha) = \mu_1 - \frac{\alpha \mu_2^2 \eta \Lambda}{2}$. We require that this quantity is greater than zero, $\delta(\alpha) > 0$; this discussion is deferred to the end of the proof.

Taking an expectation over all batches $S_0, S_1, ..., S_{k-1}$ and all history starting with w_0, and rearranging (3.21) yields

$$\mathbb{E}\left[\|\nabla F(w_k)\|^2\right] \le \frac{1}{\alpha \delta(\alpha)} \mathbb{E}[F(w_k) - F(w_{k+1})] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2\delta(\alpha)}.$$

Summing over $k = 0, ..., \tau - 1$ and dividing by τ

$$\begin{aligned} \min_{0 \le k \le \tau - 1} \mathbb{E}\left[\|\nabla F(w_k)\|^2\right] &\le \frac{1}{\tau} \sum_{k=0}^{\tau - 1} \mathbb{E}\left[\|\nabla F(w_k)\|^2\right] \\ &\le \frac{1}{\alpha \delta(\alpha) \tau} \mathbb{E}[F(w_0) - F(w_\tau)] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2\delta(\alpha)} \\ &\le \frac{1}{\alpha \delta(\alpha) \tau} [F(w_0) - \hat{F}] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2\delta(\alpha)} \\ &\le \frac{1}{\delta(\frac{c}{\sqrt{\tau}}) c \sqrt{\tau}} [F(w_0) - \hat{F}] + \frac{c \mu_2^2 \gamma^2 \Lambda}{2\delta(\frac{c}{\sqrt{\tau}}) \sqrt{\tau}}. \end{aligned}$$

The first inequality holds because the minimum value is less than the average value, and the third inequality holds because $\hat{F} \le F(x_\tau)$ (Assumption $B.3$). The last expression

comes as a result of using the definition of the step length, $\alpha = \frac{c}{\sqrt{\tau}}$. Setting

$$c = \sqrt{\frac{2(F(w_0) - \hat{F})}{\mu_2^2 \gamma^2 \Lambda}}, \tag{3.22}$$

yields the desired result.

We now comment on the quantity $\delta(\alpha)$ that first appears in (3.21), and that is required to be positive. To ensure that $\delta(\alpha) > 0$, the step length must satisfy, $\alpha < \frac{2\mu_1}{\mu_2^2 \eta \Lambda}$. Since the explicit form of the step length is $\alpha = \frac{c}{\sqrt{\tau}}$, where c is (3.22), we require that

$$\alpha = \frac{c}{\sqrt{\tau}} < \frac{2\mu_1}{\mu_2^2 \eta \Lambda}. \tag{3.23}$$

In order to ensure that (3.23) holds, we impose that

$$\tau > \frac{c^2 \mu_2^4 \eta^2 \Lambda^2}{4\mu_1^2} = \frac{(F(w_0) - \hat{F})\mu_2^2 \eta^2 \Lambda}{2\gamma^2 \mu_1^2}.$$

■

The result of Theorem 3.7 establishes a sub-linear rate of convergence, to a stationary point of F, for the multi-batch L-BFGS method on nonconvex objective functions. The result is somewhat strange as it requires a priori knowledge of τ, the total number of iteration. In practice, one would use $\alpha_k = \frac{1}{\sqrt{k}}$, which would result in a $\mathcal{O}(\frac{1}{\sqrt{k}})$ convergence rate.

4. Numerical Results

We present numerical experiments on several problems that arise in machine learning, such as logistic regression binary classification and neural network training, in order to evaluate the performance of the proposed multi-batch L-BFGS method. The experiments verify that the proposed method is robust, competitive and achieves a good balance between computation and communication in the distributed setting. In Section 4.1, we evaluate the performance of the multi-batch L-BFGS method on binary classification tasks in both the multi-batch and fault-tolerant settings. In Section 4.2, we demonstrate the performance of the multi-batch L-BFGS method on neural network training tasks, and compare against some of the state-of-the-art methods. Finally, in Section 4.3, we illustrate the strong and weak scaling properties of the multi-batch L-BFGS method.

4.1 *Logistic Regression*

In this section, we focus on logistic regression problems; the optimization problem can be stated as:

$$\min_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y^i (w^T x^i)}\right) + \frac{\sigma}{2} \|w\|^2,$$

where $(x^i, y^i)_{i=1}^n$ denote the training examples and $\sigma = \frac{1}{n}$ is the regularization parameter.

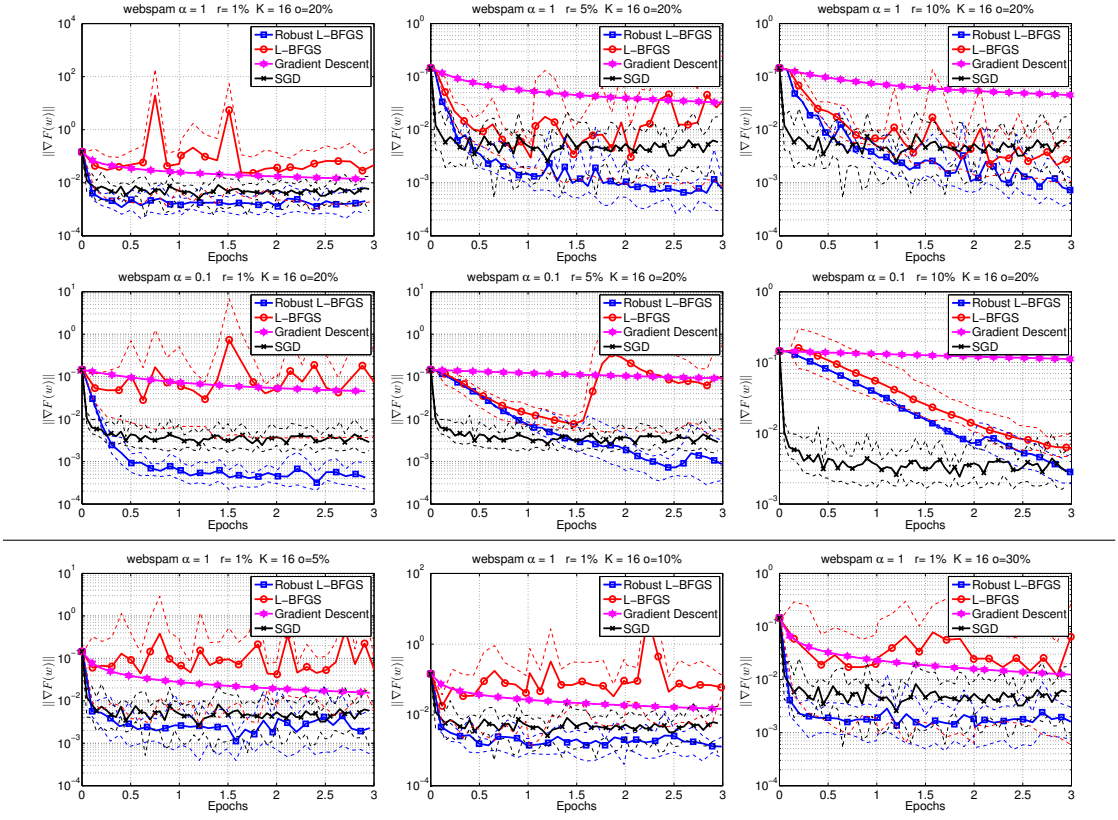


Figure 2. **webspam dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

We present numerical results that evaluate the performance of the proposed robust multi-batch L-BFGS scheme (Algorithm 1) in both the *multi-batch* (Figure 2) and *fault tolerant* (Figure 3) settings, on the `webspam` dataset[2]. We compare our proposed method (Robust L-BFGS) against three methods: (i) multi-batch L-BFGS without enforcing sample consistency (L-BFGS), where gradient differences are computed using different samples, i.e., $y_k = g_{k+1}^{S_{k+1}} - g_k^{S_k}$; (ii) multi-batch gradient descent (Gradient Descent), which is obtained by setting $H_k = I$ in Algorithm 1; and, (iii) serial SGD (SGD), where at every iteration one sample is used to compute the gradient. We run each method with 10 different random seeds, and, where applicable, report results for different batch (r) and overlap (o) sizes. In Figures 2 and 3 we show the evolution of the norm of the gradient in terms of epochs.

In the multi-batch setting, the proposed method is more stable than the standard L-BFGS method; this is especially noticeable when r is small. On the other hand, serial SGD achieves similar accuracy as the robust L-BFGS method and at a similar rate, at the cost of n communications per epoch versus $\frac{1}{r(1-o)}$ communications per epoch. Figure 2 also indicates that the robust L-BFGS method is not too sensitive to the size of the

[2]LIBSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

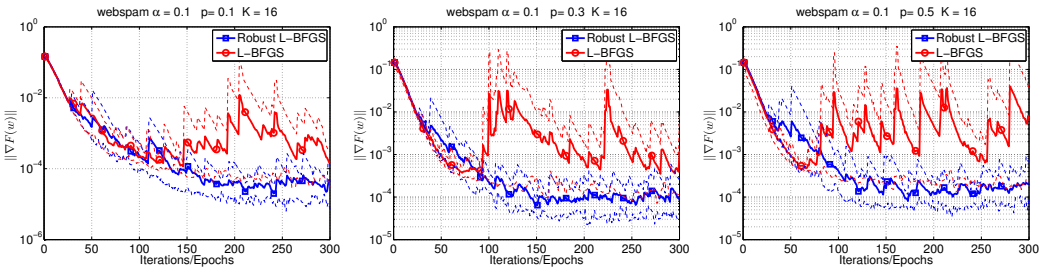


Figure 3. **webspam dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.3, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

overlap. Similar behavior was observed on other datasets, in regimes where $r \cdot o$ was not too small; see Appendix A.1.

Figure 3 shows a comparison of the proposed robust multi-batch L-BFGS method and the multi-batch L-BFGS method that does not enforce sample consistency (L-BFGS) in the presence of faults. In these experiments, p denotes the probability that a single node (MPI process) will not return a gradient evaluated on local data within a given time budget. We illustrate the performance of the methods for $\alpha = 0.1$ and $p \in \{0.1, 0.3, 0.5\}$. We observe that the robust implementation is not affected much by the failure probability p. Similar behavior was observed on other datasets; see Appendix A.2.

4.2 *Neural Networks*

In this section, we study the performance of the multi-batch L-BFGS method[3] on Neural Network tasks, on the MNIST and CIFAR10/CIFAR100 datasets[4]. Table 1 summarizes the network architectures that we used; see Appendix B for more details. The first problem is convex; all other problems are nonconvex.

Table 1. Structure of Neural Networks.

Network	Type	# of layers	d	Ref.
MNIST MLC	fully connected (FC)	1	7.8k	[39]
MNIST DNN (SoftPlus)	conv+FC	4	1.1M	[1]
MNIST DNN (ReLU)	conv+FC	4	1.1M	[1]
CIFAR10 LeNet	conv+FC	5	62.0k	[39]
CIFAR10 VGG11	conv+batchNorm+FC	29	9.2M	[64]
CIFAR100 VGG11	conv+batchNorm+FC	29	9.2M	[64]

MLC = Multiclass Linear Classifier

We implemented our algorithm in PyTorch [55] and compare against popular and readily available algorithms: (i) SGD [60], and (ii) Adam [35]. We denote our proposed method as LBFGS in the figures in this section. Note, we implemented our method with the *cautious updating* strategy, and For each method, we conducted a grid search to find the best learning rate $\alpha \in \{2^0, 2^{-1}, \ldots, 2^{-10}\}$, and also investigated the effect of different batch sizes $|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$; see Appendix B for detailed experiments with all batch sizes. For the multi-batch L-BFGS method we also

[3]Code available at: https://github.com/OptMLGroup/Multi-Batch_L-BFGS.

[4]MNIST available at: http://yann.lecun.com/exdb/mnist/. CIFAR10/CIFAR100 available at: https://www.cs.toronto.edu/~kriz/cifar.html.

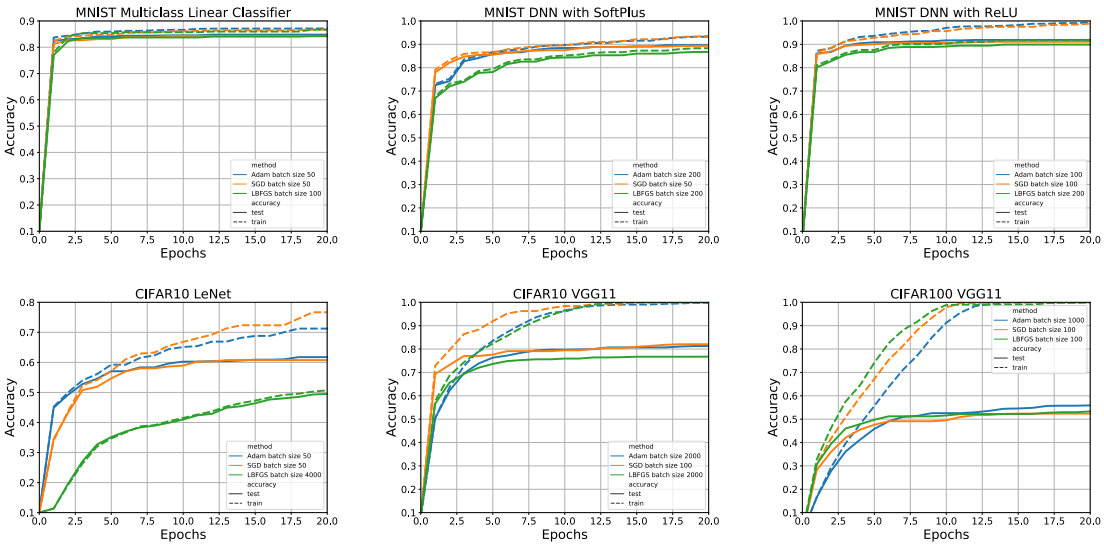


Figure 4. Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for the best parameter setting for each method and problem. Top right: MNIST Multiclass Linear Classifier; Top middle: MNIST DNN (SoftPlus); Top Right: MNIST DNN (ReLU); Bottom left: CIFAR10 LeNet; Bottom middle: CIFAR10 VGG11; Bottom right: CIFAR100 VGG11.

investigated the effect of history length $m \in \{1, 2, 5, 10, 20\}$. The overlap used in our proposed method was 20% of the batch, $o = 0.20$.

The authors in [34] observed that the widely used Barzilai-Borwein-type scaling $\frac{s_k^T y_k}{y_k^T y_k} I$ of the initial Hessian approximation may lead to quasi-Newton updates that are not stable when small batch sizes are employed, especially for deep neural training tasks, and as such propose an Agadrad-like scaling of the initial BFGS matrix. To obviate this instability, we implement a variant of the multi-batch L-BFGS method (LBFGS2) in which we scale the initial Hessian approximations as αI. We ran experiments with both scaling strategies and the overall results were similar. Therefore, in the figures in this section we only show results for the latter strategy.

Figure 4 illustrates the evolution of the training accuracy (dashed lines) and testing accuracy (solid lines) for the best parameter settings for each method over the first 20 epochs of training. By best parameter settings we mean the run that achieved the highest training accuracy within 20 epochs. One can make several observations. First, it appears that the multi-batch L-BFGS method is competitive with the first-order methods on all training problems, except for CIFAR10 LeNet, in terms of both training and testing accuracy. Second, for almost all problems, the best runs of the multi-batch L-BFGS method use larger batch sizes than the first-order methods. Third, the benefits of incorporating second-order information are not as apparent in these nonconvex problems as compared to the problems presented in Section 4.1. We attribute this to two things: (1) nonconvex problems are hard, and (2) quasi-Newton methods are better are capturing curvature information for convex problems.

We now investigate the effect of the batch size. In Figure 5 we show the evolution of the training/testing accuracy for different batch sizes $|S| \in \{50, 500, 4000\}$ for three of the problems. For a complete set of results, see Appendix B. Overall, one can observe that for small batch sizes, the multi-batch L-BFGS variants perform worse than the first order methods. However, when large batches are employed (a regime that is favorable foe GPU computing), the multi-batch L-BFGS method performs on par with the other

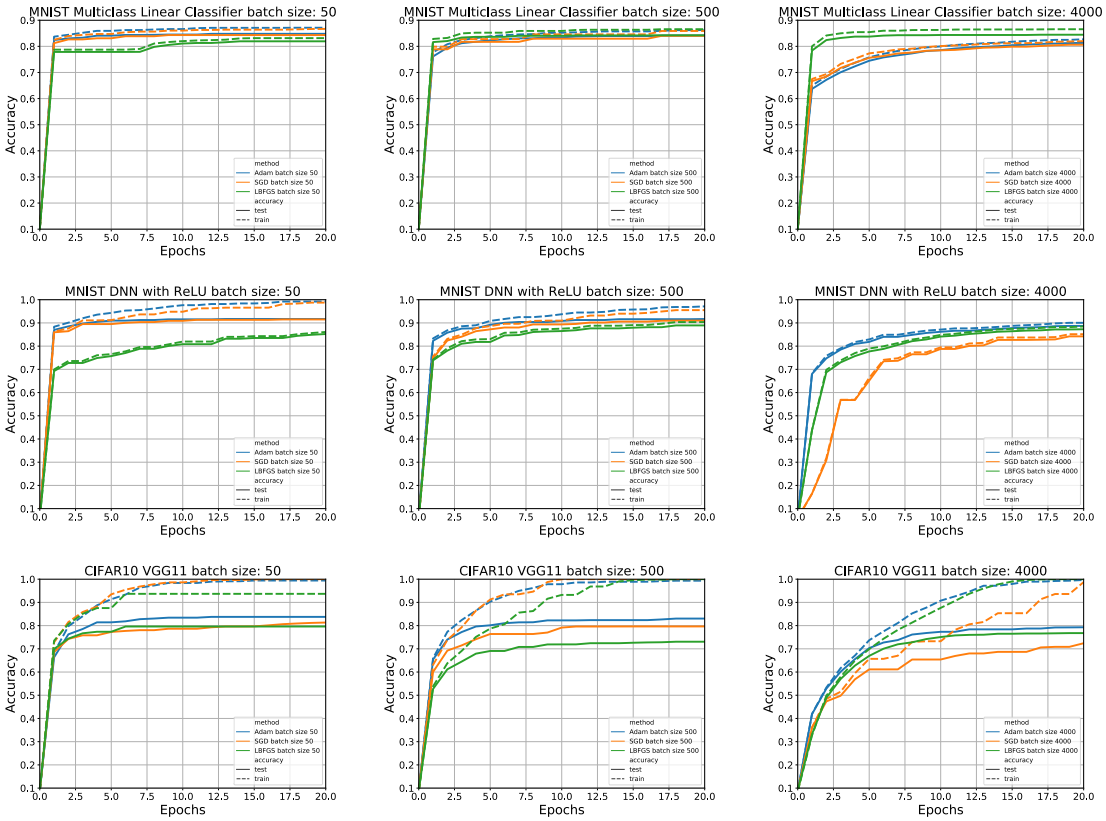


Figure 5. Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 500, 4000\}$). Top row: MNIST Multiclass Linear Classifier; Middle row: MNIST DNN (ReLU); Bottom row: CIFAR10 VGG11.

methods. Moreover, it appears that the performance of the multi-batch L-BFGS methods is less affected by the size of the batch.

In order to understand why the multi-batch L-BFGS method does not perform well for small batches, we looked at two diagnostic measures: (i) the angle between the true gradient curvature vector y_d and subsampled gradient curvature vector y_s $(\frac{\langle y_s, y_d \rangle}{\|y_s\|\|y_d\|})$; and (ii) the ratio of subsampled gradient curvature vector to true gradient curvature vector $(\frac{y_s}{y_d})$. These measures indicate how informative the curvature information captured by the multi-batch L-BFGS method really is. Values close to 1 (dashed red lines) are ideal for both measures. We chose 3 different points (the starting point, a point after 3 epochs of Adam, and a point after 10 epochs of Adam). From those points we took a gradient descent step with sufficiently small step length, and computed the true gradient curvature vector (y_d). We also computed 100 different stochastic variants of the gradient curvature vector (y_s) using different batch sizes $(|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\})$, and calculated the values of the two metrics. We illustrate the results in Figure 6; see Appendix B for more results. Several observations can be drawn from this figure. First, not surprisingly, the metrics improve (get closer to 1) as the batch size increases. Second, for the convex case, the metrics perform as expected both close and far from the solution; as a result (sufficiently) good curvature information is captured and the method performs well. On the other hand, for the nonconvex problems, the metrics indicate that, especially for small batch sizes, the curvature information captured can be terrible.

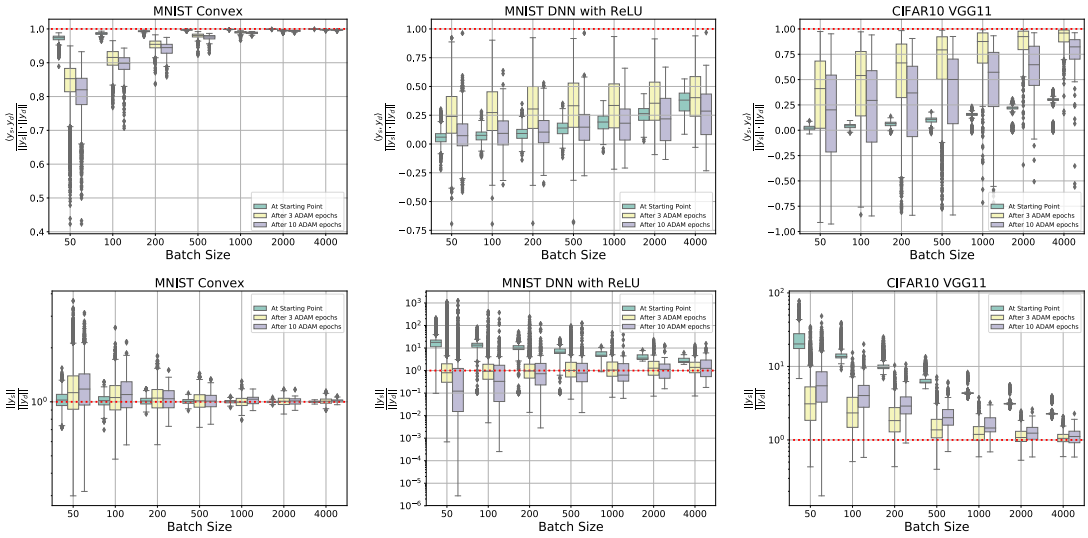


Figure 6. Multi-batch L-BFGS diagnostic metrics for different batch sizes $|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$. Top row: angle between the true gradient curvature vector y_d and subsampled gradient curvature vector y_s ($\frac{\langle y_s, y_d \rangle}{\|y_s\| \|y_d\|}$); Bottom row: ratio of subsampled gradient curvature vector to true gradient curvature vector ($\frac{y_s}{y_d}$). Left column: MNIST Multiclass Linear Classifier; Middle column: MNIST DNN (ReLU); Right column: CIFAR10 VGG11.

We should note that using a large batch size is not a bottleneck for current high-performance computing hardware, on the contrary, using small batch sizes leads to under-utilization of computational hardware and in fact hinders the ability to parallelize the methods. Figure 7 illustrates this phenomenon; we show the average computational time of a gradient evaluation (over 1,000 evaluations) for different batch sizes ranging from 1 to 4096 relative to the computational time of the computation of the gradient using a single data point. The computation was performed on an NVIDIA Tesla K80 GPU using PyTorch. It is clear that for the *Multiclass Linear classification* task, the compute time of a single gradient is roughly the same as the compute time of a gradient based on a batch of size 1024, whereas for the larger training tasks, the compute time of the gradients appear constant up to batch sizes of roughly 128.

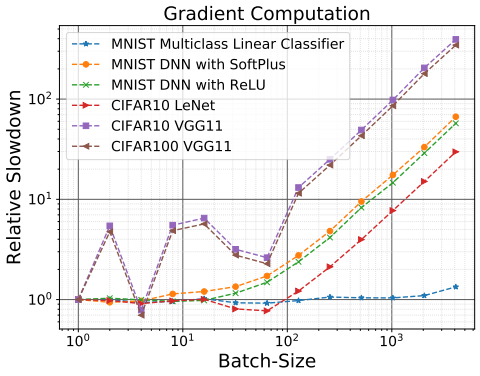


Figure 7. Relative slow-down of computational time to compute gradients for different batch-sizes compared to the computational time to compute gradients with batch-size 1.

4.3 *Scaling of the Multi-Batch L-BFGS Implementation*

In this section, we study the strong and weak scaling properties of the robust multi-batch L-BFGS method on artificial data. For various values of batch size (r) and nodes (K), we measure the time needed to compute a gradient (Gradient) and the time needed to compute and communicate the gradient (Gradient+C), as well as, the time needed to compute the L-BFGS direction (L-BFGS) and the associated communication overhead (L-BFGS+C).

Strong Scaling

Figure 8 depicts the strong scaling properties of the multi-batch L-BFGS method, for different batch sizes (r) and nodes ($K = 1, 2, ..., 128$). For this task, we generate a dataset with $n = 10^7$ samples and $d = 10^4$ dimensions, where each sample has 160 randomly chosen non-zero elements (dataset size 24GB). One can observe that as the number of nodes (K) is increased, the compute times for the gradient and the L-BFGS direction decrease. However, when communication time is considered, the combined cost increases slightly as K is increased. Notice that for large K, even when $r = 10\%$ (i.e., 10% of all samples processed in one iteration, ∼18MB of data), the amount of local work is not sufficient to overcome the communication cost.

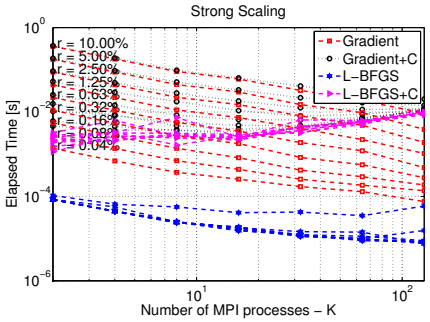


Figure 8. Strong scaling of robust multi-batch L-BFGS on a problem with artificial data; $n = 10^7$ and $d = 10^4$. Each sample has 160 non-zero elements (dataset size 24GB).

Weak Scaling – Fixed Problem Dimension, Increasing Data Size

In order to illustrate the weak scaling properties of the algorithm, we generate a data-matrix $X \in \mathbb{R}^{n \times d}$ ($n = 10^7$, $d = 10^4$), and compute the gradient and the L-BFGS direction on a shared cluster with different number of MPI processes ($K = 1, 2, ..., 128$). Each sample has $10 \cdot K$ non-zero elements, thus for any K the size of local problem is roughly 1.5GB (for $K = 128$ size of data 192GB). Effectively, the dataset size (n) is held fixed, but the sparsity of the data decreases as more MPI processes are used. The compute time for the gradient is almost constant, this is because the amount of work per MPI process (rank) is almost identical; see Figure 9. On the other hand, because we are using a Vector-Free L-BFGS implementation [19] for computing the L-BFGS direction, the amount of time needed for each node to compute the L-BFGS direction decreases as K is increased. However, increasing K does lead to larger communication overhead, and as such the overall time needed to compute and communicate the L-BFGS direction increases slightly as K is increased. For $K = 128$ (192GB of data) and $r = 10\%$, almost 20GB of data are processed per iteration in less than 0.1 seconds, which implies that one epoch would take around 1 second.

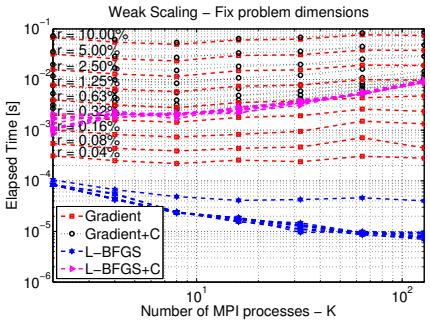


Figure 9. Weak scaling of robust multi-batch L-BFGS on a problem with artificial data; $n = 10^7$ and $d = 10^4$. Each sample has $10 \cdot K$ non-zero elements (size of local problem 1.5GB).

Increasing Problem Dimension, Fixed Data Size and K

In this experiment, we investigate the effect of a change in the dimension (d) of the problem on the computation of the gradient and the L-BFGS direction. We fix the size of data (29GB) and the number of MPI processes ($K = 8$), and generate data with $n = 10^7$ samples, where each sample has 200 non-zero elements. Figure 10 shows that increasing the dimension d has a mild effect on the computation time of the gradient, while the effect on the time needed to compute the L-BFGS direction is more apparent. However, if communication time is taken into consideration, the time required for the gradient computation and the L-BFGS direction computation increase as d is increased.

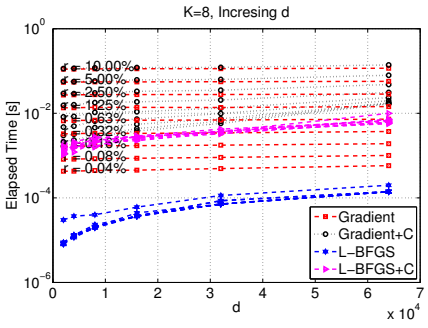


Figure 10. Scaling of robust multi-batch L-BFGS on a problem with artificial data; $n = 10^7$, increasing d and $K = 8$ MPI processes. Each sample had 200 non-zero elements (dataset size 29GB).

5. Final Remarks

In this paper, we assumed that sample consistency is not possible (*fault-tolerant* setting) or desirable (*multi-batch* setting), and described a novel and robust variant of the L-BFGS method designed to deal with two adversarial situations. The success of the algorithm relies on the fact that gradient differences need not be computed on the full batch, rather a small subset can be used alleviating the need for double function evaluations while still maintaining useful curvature information. The method enforces a small degree of control in the sampling process and avoids the pitfalls of using inconsistent gradient differences by performing quasi-Newton updating on the overlap between consecutive samples.

Our numerical results indicate that provided the overlap is not too small, the proposed method is efficient in practice on machine learning tasks such as binary classification logistic regression and neural network training. The experiments presented in this paper show that the empirical performance of the method matches that predicted by the theory for both strongly convex and nonconvex functions. Specifically, in the strongly convex case the multi-batch L-BFGS method with a constant step length converges to a neighborhood of the solution at a linear rate, and in the nonconvex case the iterates produced by the multi-batch L-BFGS method converge to a neighborhood of a stationary point.

Of course, the development, both theoretical and practical, of stochastic quasi-Newton methods is far from complete, and there are many interesting directions that can and should be investigated. Theoretical analysis that would suggest the batch size and overlap size would be of great interest in practice. Moreover, an investigation of the multi-batch L-BFGS method that employs variance reduced gradients in lieu of the stochastic gradients could have both theoretical and practical advantages.

References

[1] *Pytorch examples*, https://github.com/pytorch/examples/blob/master/mnist/main.py, accessed: 2019-03-29.

[2] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, *A reliable effective terascale linear learning system*, Journal of Machine Learning Research 15 (2014), pp. 1111–1133.

[3] A.S. Berahas, R. Bollapragada, and J. Nocedal, *An investigation of newton-sketch and subsampled newton methods*, arXiv preprint arXiv:1705.06211 (2017).
[4] A.S. Berahas, M. Jahani, and M. Takáč, *Quasi-newton methods for deep learning: Forget the past, just sample*, arXiv preprint arXiv:1901.09997 (2019).
[5] A.S. Berahas, J. Nocedal, and M. Takáč, *A Multi-Batch L-BFGS Method for Machine Learning*, in *Advances in Neural Information Processing Systems 29*, 2016, pp. 1055–1063.
[6] A.S. Berahas and M. Takáč, *A multi-batch L-BFGS method with variance-reduced gradients: Theory and experiments*, arXiv preprint (2017).
[7] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and distributed computation: Numerical methods*, Vol. 23, Prentice Hall Englewood Cliffs, NJ, 1989.
[8] R. Bollapragada, R. Byrd, and J. Nocedal, *Exact and inexact subsampled Newton methods for optimization*, arXiv preprint arXiv:1609.08502 (2016).
[9] R. Bollapragada, R. Byrd, and J. Nocedal, *Adaptive sampling strategies for stochastic optimization*, SIAM Journal on Optimization 28 (2018), pp. 3312–3343.
[10] R. Bollapragada, D. Mudigere, J. Nocedal, H.J.M. Shi, and P.T.P. Tang, *A Progressive Batching L-BFGS Method for Machine Learning*, in *International Conference on Machine Learning*, 2018, pp. 619–628.
[11] A. Bordes, L. Bottou, and P. Gallinari, *SGD-QN: Careful quasi-Newton stochastic gradient descent*, Journal of Machine Learning Research 10 (2009), pp. 1737–1754.
[12] L. Bottou, F.E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, Siam Review 60 (2018), pp. 223–311.
[13] L. Bottou and Y. Le Cun, *Large scale online learning*, in *Advances in Neural Information Processing Systems 16*, 2003, pp. 217–224.
[14] O. Bousquet and L. Bottou, *The tradeoffs of large scale learning*, in *Advances in Neural Information Processing Systems 20*, 2007, pp. 161–168.
[15] C.G. Broyden, *Quasi-Newton methods and their application to function minimisation*, Mathematics of Computation 21 (1967), pp. 368–381.
[16] R.H. Byrd, G.M. Chin, W. Neveitt, and J. Nocedal, *On the use of stochastic Hessian information in optimization methods for machine learning*, SIAM Journal on Optimization 21 (2011), pp. 977–995.
[17] R.H. Byrd, G.M. Chin, J. Nocedal, and Y. Wu, *Sample size selection in optimization methods for machine learning*, Mathematical Programming 134 (2012), pp. 127–155.
[18] R.H. Byrd, S.L. Hansen, J. Nocedal, and Y. Singer, *A stochastic quasi-Newton method for large-scale optimization*, SIAM Journal on Optimization 26 (2016), pp. 1008–1031.
[19] W. Chen, Z. Wang, and J. Zhou, *Large-scale L-BFGS using MapReduce*, in *Advances in Neural Information Processing Systems*, 2014, pp. 1332–1340.
[20] F. Curtis, *A self-correcting variable-metric algorithm for stochastic optimization*, in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 632–641.
[21] Y.H. Dai, *Convergence properties of the BFGS algoritm*, SIAM Journal on Optimization 13 (2002), pp. 693–701.
[22] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, *Distributed deep learning using synchronous stochastic gradient descent*, arXiv preprint arXiv:1602.06709 (2016).
[23] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q.V. Le, *et al.*, *Large scale distributed deep networks*, in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1223–1231.
[24] A. Defazio, F. Bach, and S. Lacoste-Julien, *SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives*, in *Advances in Neural Information Processing Systems 27*, 2014, pp. 1646–1654.
[25] R. Fletcher, *A new approach to variable metric algorithms*, The computer journal 13 (1970), pp. 317–322.
[26] M.P. Friedlander and M. Schmidt, *Hybrid deterministic-stochastic methods for data fitting*, SIAM Journal on Scientific Computing 34 (2012), pp. A1380–A1405.
[27] S. Ghadimi and G. Lan, *Stochastic first-and zeroth-order methods for nonconvex stochastic programming*, SIAM Journal on Optimization 23 (2013), pp. 2341–2368.
[28] D. Goldfarb, *A family of variable-metric methods derived by variational means*, Mathematics of computation 24 (1970), pp. 23–26.
[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org.
[30] R. Gower, D. Goldfarb, and P. Richtarik, *Stochastic Block BFGS: Squeezing More Curvature out*

of Data, in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 1869–1878.

[31] M. Hardt, B. Recht, and Y. Singer, *Train faster, generalize better: Stability of stochastic gradient descent*, in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 1225–1234.

[32] X. He and M. Takáč, *Dual free SDCA for empirical risk minimization with adaptive probabilities*, OptML@NIPS 2015, arXiv:1510.06684 (2015).

[33] R. Johnson and T. Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, in *Advances in Neural Information Processing Systems 26*, 2013, pp. 315–323.

[34] N.S. Keskar and A.S. Berahas, *adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs*, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2016, pp. 1–16.

[35] D.P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[36] J. Konečný and P. Richtárik, *Semi-stochastic gradient descent methods*, Frontiers in Applied Mathematics and Statistics 3 (2017), p. 9.

[37] J. Konečný, J. Liu, P. Richtárik, and M. Takáč, *Mini-batch semi-stochastic gradient descent in the proximal setting*, IEEE Journal of Selected Topics in Signal Processing 10 (2016), pp. 242–255.

[38] R. Leblond, F. Pedregosa, and S. Lacoste-Julien, *ASAGA: Asynchronous parallel SAGA*, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 46–54.

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

[40] D.H. Li and M. Fukushima, *A modified BFGS method and its global convergence in nonconvex minimization*, Journal of Computational and Applied Mathematics 129 (2001), pp. 15–35.

[41] D.H. Li and M. Fukushima, *On the global convergence of the BFGS method for nonconvex unconstrained optimization problems*, SIAM Journal on Optimization 11 (2001), pp. 1054–1064.

[42] H. Lin, J. Mairal, and Z. Harchaoui, *A universal catalyst for first-order optimization*, in *Advances in Neural Information Processing Systems 28*, 2015, pp. 3384–3392.

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

[44] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M.I. Jordan, *Perturbed iterate analysis for asynchronous stochastic optimization*, arXiv preprint arXiv:1507.06970 (2015).

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

[46] W.F. Mascarenhas, *The BFGS method with exact line searches fails for non-convex objective functions*, Mathematical Programming 99 (2004), pp. 49–61.

[47] A. Mokhtari and A. Ribeiro, *Global convergence of online limited memory BFGS*, Journal of Machine Learning Research 16 (2015), pp. 3151–3181.

[48] A. Nedić and D. Bertsekas, *Convergence rate of incremental subgradient algorithms*, in *Stochastic Optimization: Algorithms and Applications*, Springer, 2001, pp. 223–264.

[49] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*, Vol. 87, Springer Science & Business Media, 2013.

[50] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q.V. Le, and A.Y. Ng, *On optimization methods for deep learning*, in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 265–272.

[51] L. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, *SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient*, in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[52] L.M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, *Stochastic recursive gradient algorithm for nonconvex optimization*, arXiv preprint arXiv:1705.07261 (2017).

[53] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Mathematics of computation 35 (1980), pp. 773–782.

[54] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., Springer New York, 1999.

[55] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in PyTorch*, in *NIPS-W*, 2017.

[56] M.J. Powell, *Some global convergence properties of a variable metric algorithm for minimization without exact line searches*, Nonlinear programming 9 (1976), pp. 53–72.

[57] M.J. Powell, *Algorithms for nonlinear constraints that use lagrangian functions*, Mathematical programming 14 (1978), pp. 224–248.

[58] B. Recht, C. Re, S. Wright, and F. Niu, *Hogwild: A lock-free approach to parallelizing stochastic gradient descent*, in *Advances in Neural Information Processing Systems 24*, 2011, pp. 693–701.

[59] S.J. Reddi, A. Hefny, S. Sra, B. Poczos, and A.J. Smola, *On variance reduction in stochastic gradient descent and its asynchronous variants*, in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2647–2655.

[60] H. Robbins and S. Monro, *A stochastic approximation method*, The annals of mathematical statistics (1951), pp. 400–407.

[61] M. Schmidt, N. Le Roux, and F. Bach, *Minimizing finite sums with the stochastic average gradient*, Mathematical Programming (2016), pp. 1–30.

[62] N.N. Schraudolph, J. Yu, and S. Günter, *A Stochastic Quasi-Newton Method for Online Convex Optimization.*, in *Proceedings of the 10th International Conference on Artificial Intelligence and Statistics*, Vol. 7, 2007, pp. 436–443.

[63] D.F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Mathematics of computation 24 (1970), pp. 647–656.

[64] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556 (2014).

[65] M. Takáč, A. Bijral, P. Richtarik, and N. Srebro, *Mini-Batch Primal and Dual Methods for SVMs*, in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 1022–1030.

[66] J. Tsitsiklis, D. Bertsekas, and M. Athans, *Distributed asynchronous deterministic and stochastic gradient optimization algorithms*, IEEE transactions on automatic control 31 (1986), pp. 803–812.

[67] X. Wang, S. Ma, D. Goldfarb, and W. Liu, *Stochastic quasi-Newton methods for nonconvex stochastic optimization*, arXiv preprint arXiv:1607.01231 (2015).

[68] Y. Zhang and X. Lin, *DiSCO: Distributed optimization for self-concordant empirical loss*, in *Proceedings of the 32th International Conference on Machine Learning*, 2015, pp. 362–370.

[69] M. Zinkevich, M. Weimer, L. Li, and A.J. Smola, *Parallelized stochastic gradient descent*, in *Advances in Neural Information Processing Systems 28*, 2010, pp. 2595–2603.

Appendix A. Extended Numerical Results - Real Datasets - Logistic Regression

In this section, we present further numerical results on binary classification logistic regression problems, on the datasets listed in Table A1, in both the multi-batch and fault-tolerant settings. Note, that some of the datasets are too small, and thus, there is no reason to run them on a distributed platform; however, we include them as they are part of the standard benchmarking datasets.

Table A1. Datasets and basic statistics. All datasets are available at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

Dataset	n	d	Size (MB)	K
ijcnn	91,701	22	14	4
cov	581,012	54	68	4
news20	19,996	1,355,191	134	4
rcvtest	677,399	47,236	1,152	16
url	2,396,130	3,231,961	2,108	16
kdda	8,407,752	20,216,830	2,546	16
kddb	19,264,097	29,890,095	4,894	16
webspam	350,000	16,609,143	23,866	16
splice-site	50,000,000	11,725,480	260,705	16

We focus on logistic regression classification; the objective function is given by

$$\min_{w\in\mathbb{R}^d} F(w) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1+e^{-y^i(w^T x^i)}\right) + \frac{\sigma}{2}\|w\|^2,$$

where $(x^i, y^i)_{i=1}^n$ denote the training examples and $\sigma = \frac{1}{n}$ is the regularization parameter.

A.1 *Extended Numerical Results - Multi-Batch Setting*

For the experiments in this section (Figures A1-A9), we ran four methods:

- (Robust L-BFGS) robust multi-batch L-BFGS (Algorithm 1),
- (L-BFGS) multi-batch L-BFGS without enforcing sample consistency; gradient differences are computed using different samples, i.e., $y_k = g_{k+1}^{S_{k+1}} - g_k^{S_k}$,
- (Gradient Descent) multi-batch gradient descent; obtained by setting $H_k = I$ in Algorithm 1,
- (SGD) serial SGD; at every iteration one sample is used to compute the gradient.

In Figures A1-A9 we show the evolution of $\|\nabla F(w)\|$ for different step lengths α, and for various batch ($|S| = r \cdot n$) and overlap ($|O| = o \cdot |S|$) sizes. Each Figure (A1-A9) consists of 10 plots that illustrate the performance of the methods with the following parameters:

- Top 3 plots: $\alpha = 1$, $o = 20\%$ and $r = 1\%, 5\%, 10\%$
- Middle 3 plots: $\alpha = 0.1$, $o = 20\%$ and $r = 1\%, 5\%, 10\%$
- Bottom 4 plots: $\alpha = 1$, $r = 1\%$ and $o = 5\%, 10\%, 20\%, 30\%$

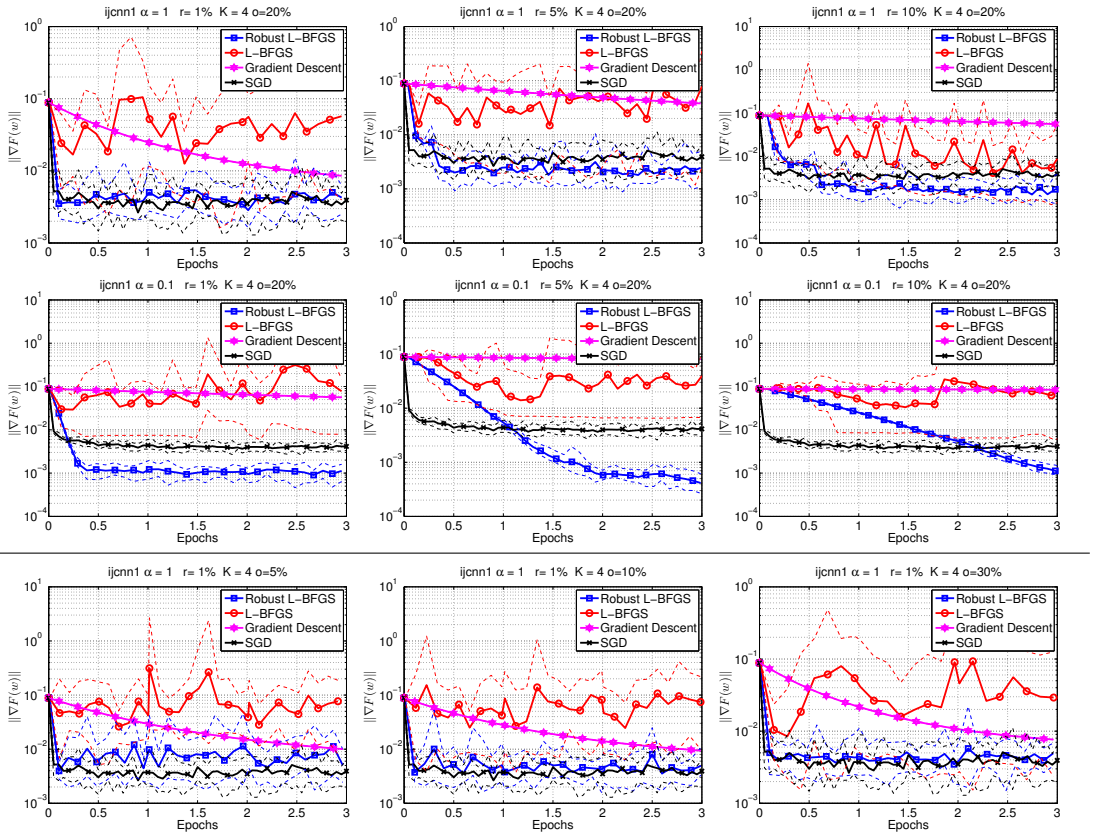


Figure A1. **ijcnn1 dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.

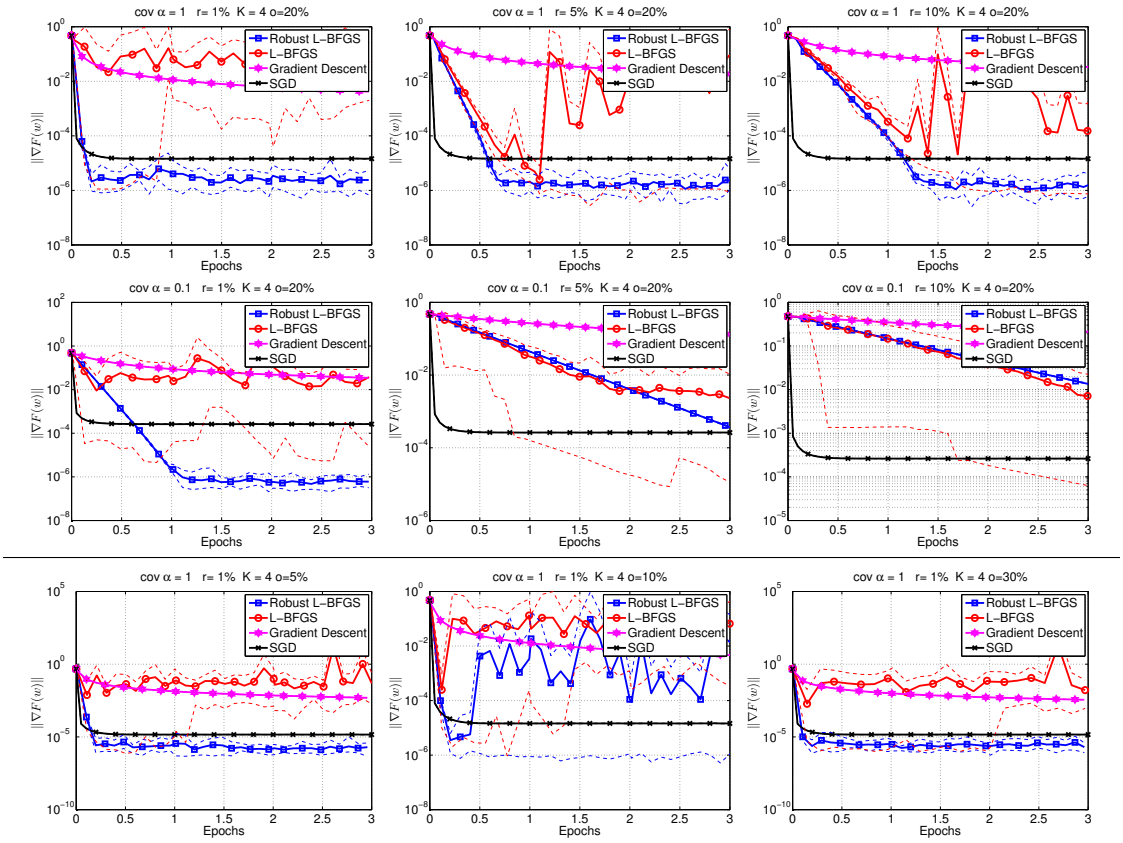


Figure A2. **cov dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.

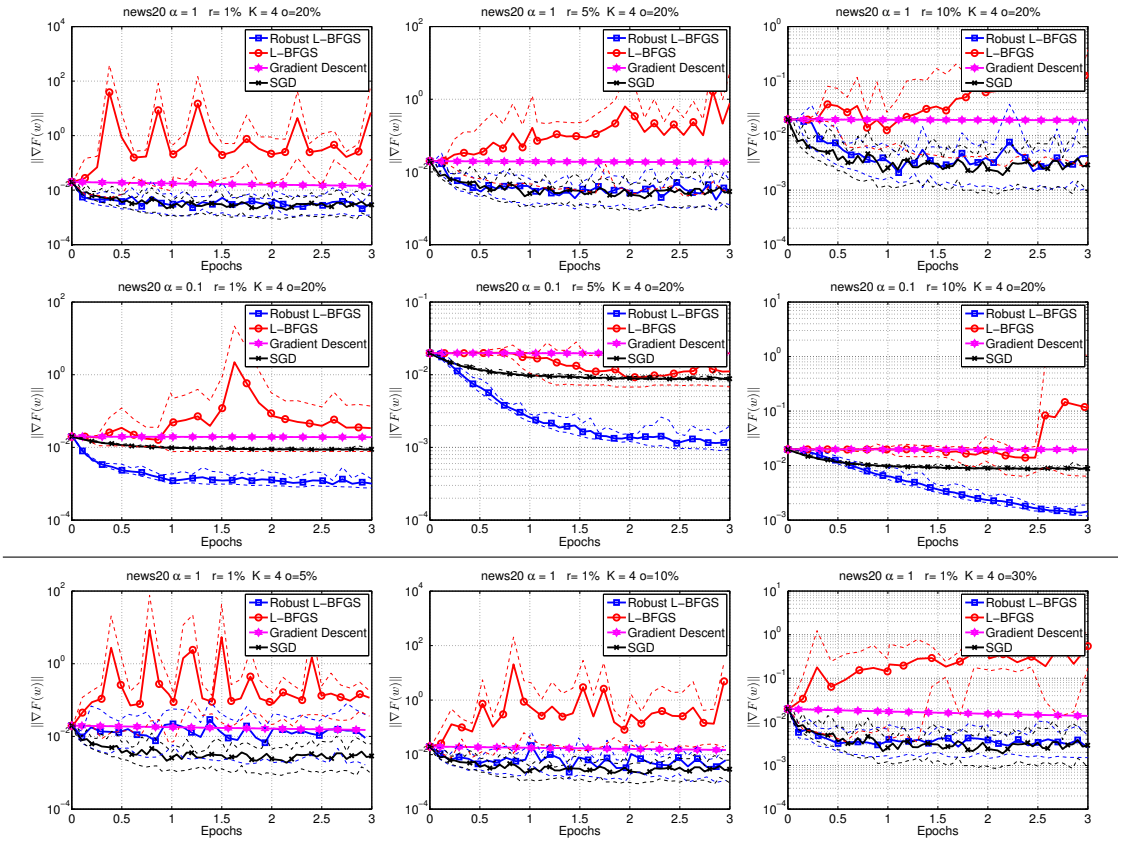


Figure A3. **news20 dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.

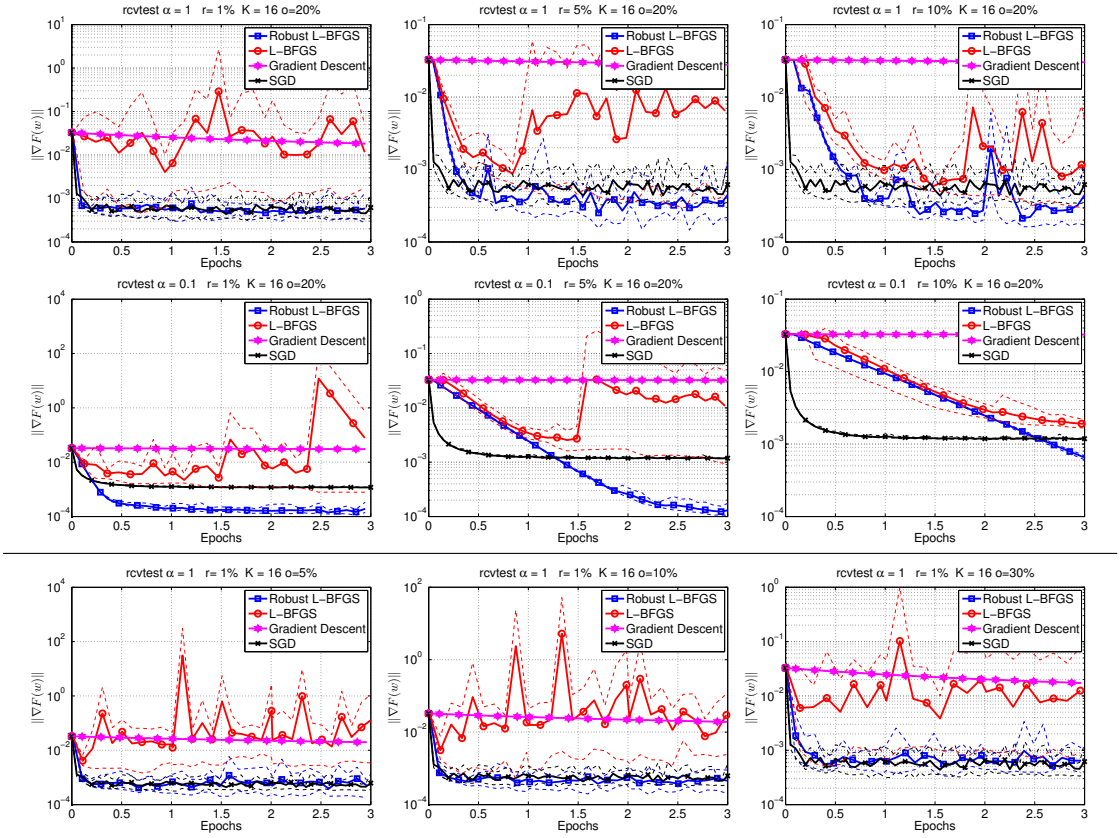


Figure A4. **rcvtest dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

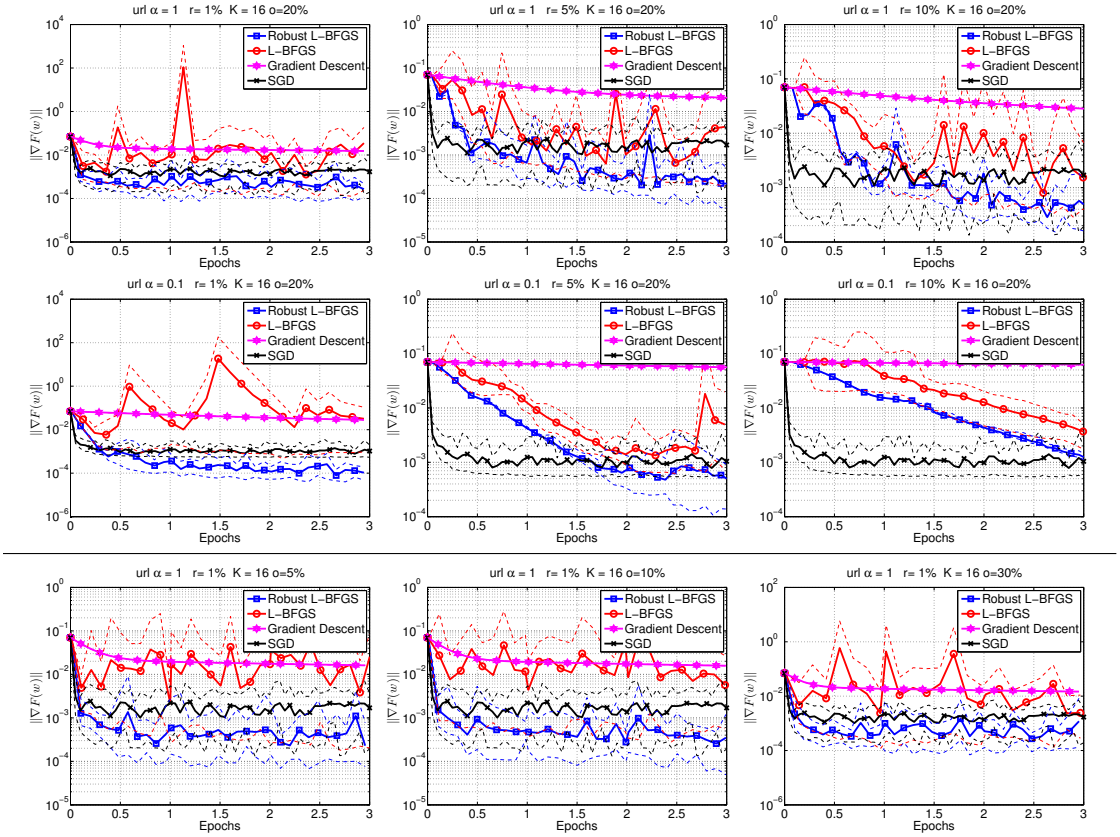


Figure A5. **url dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

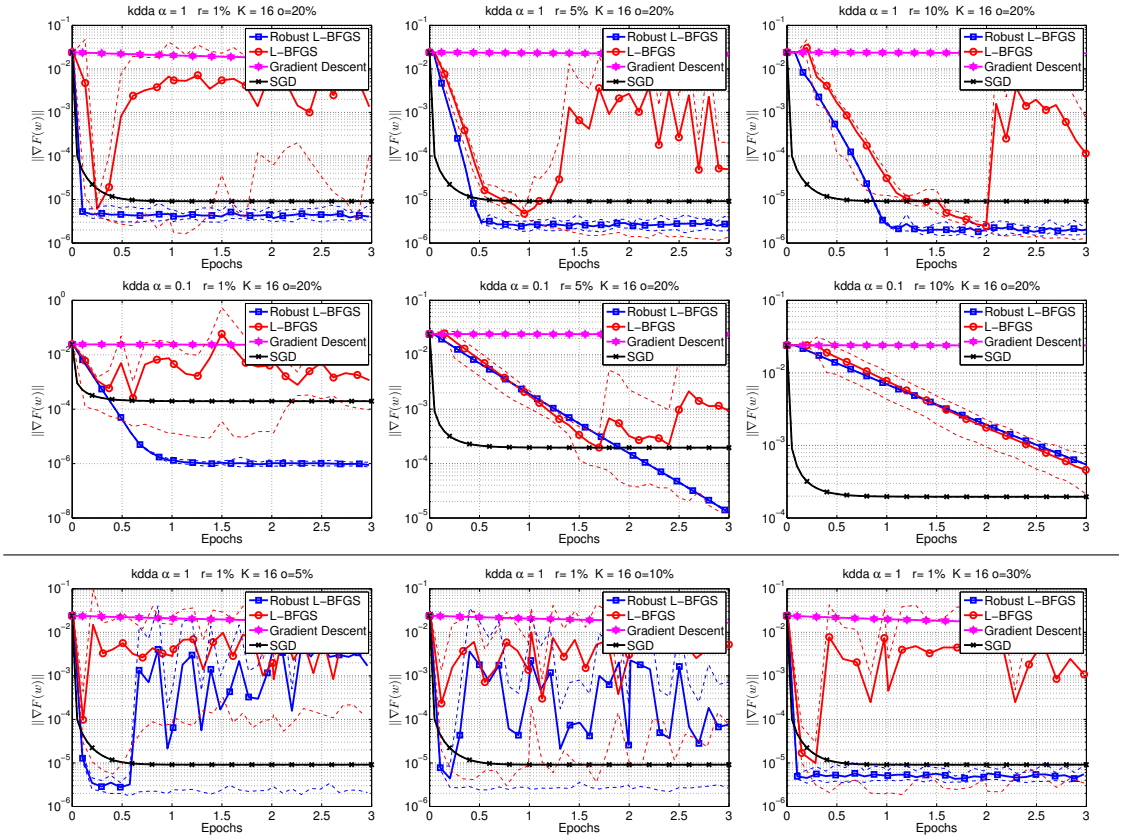


Figure A6. **kdda dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

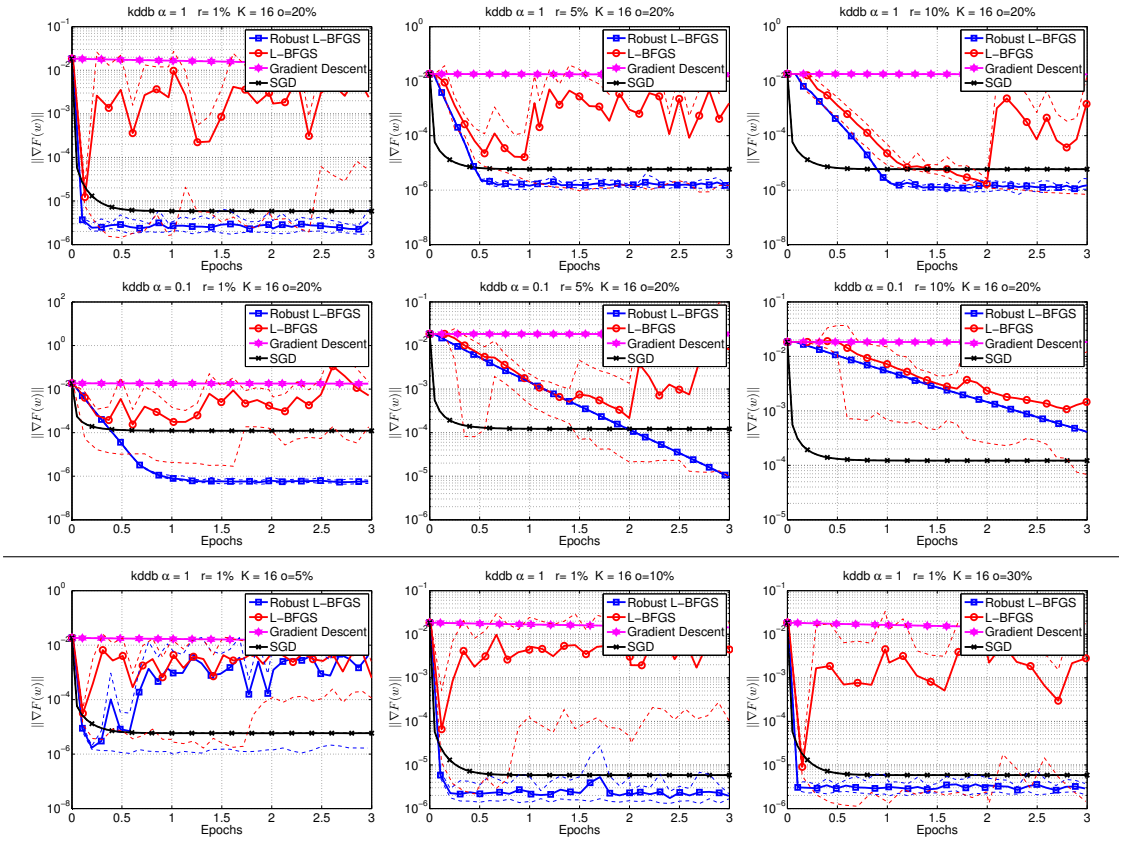


Figure A7. **kddb dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

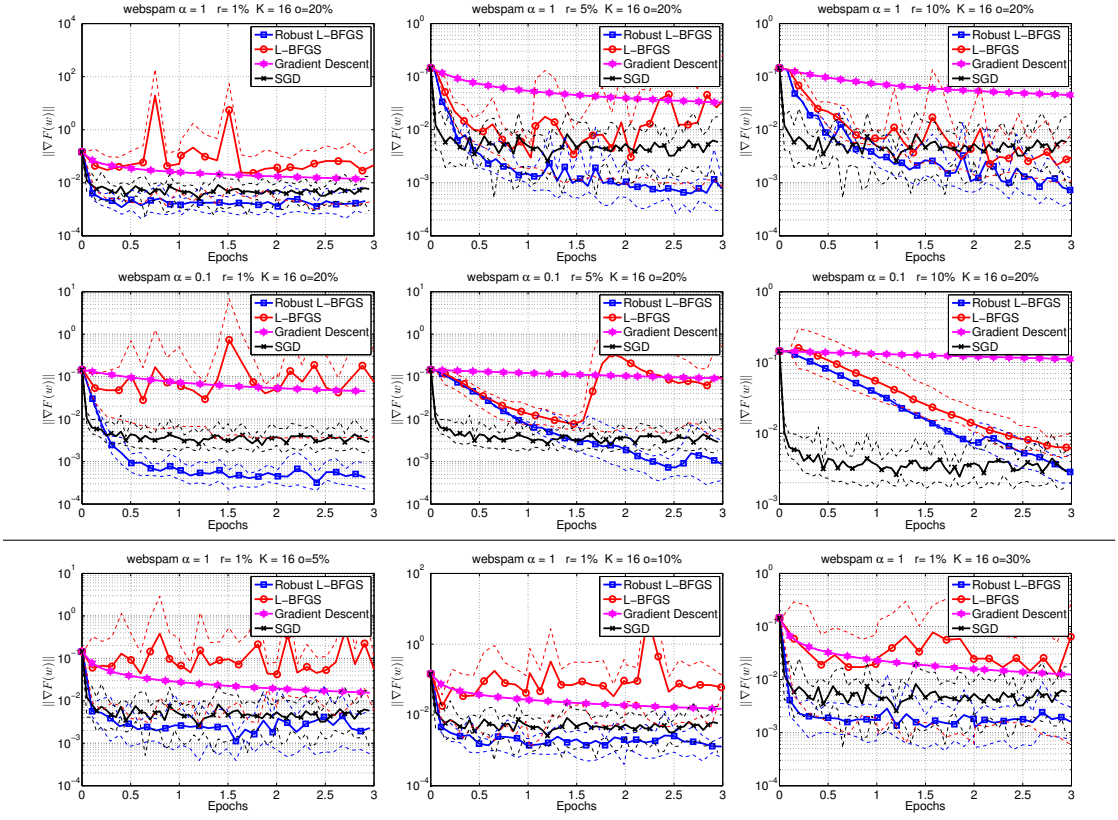


Figure A8. **webspam dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

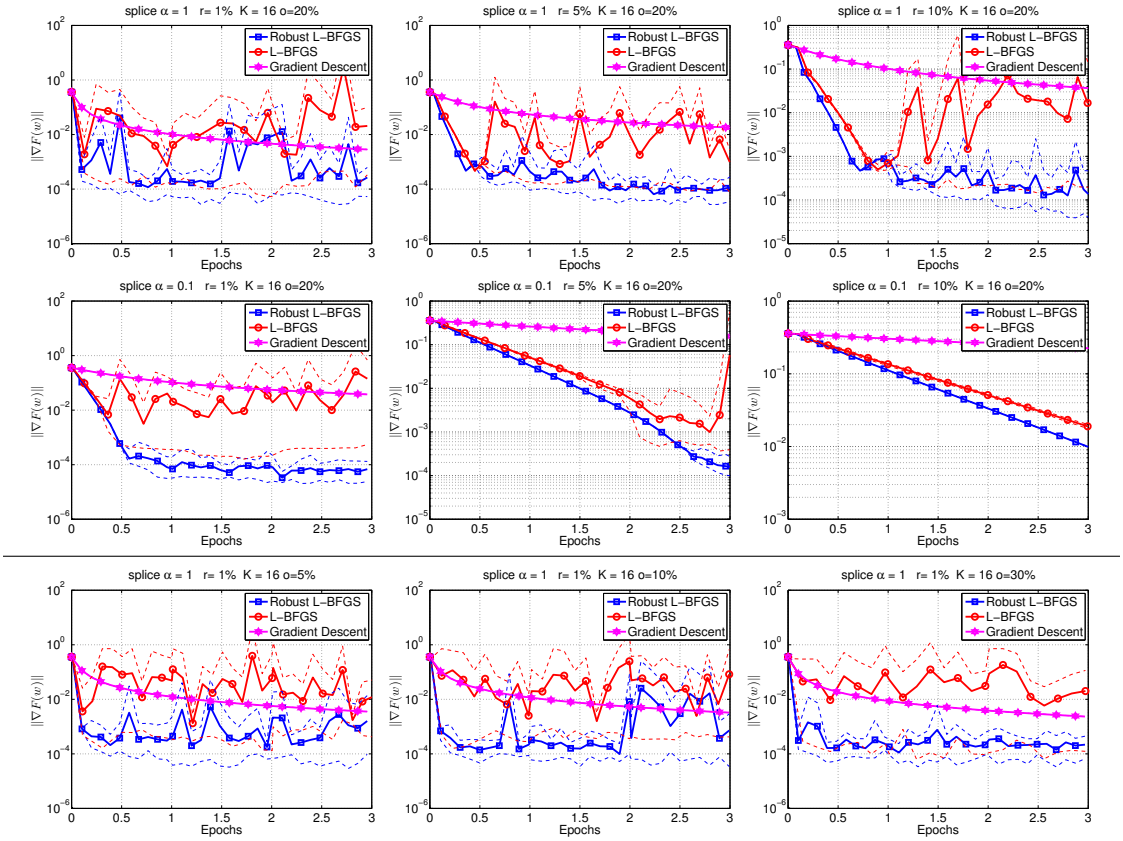


Figure A9. **splice-cite dataset**. Comparison of Robust L-BFGS, L-BFGS (multi-batch L-BFGS without enforcing sample consistency), Gradient Descent (multi-batch Gradient method) and SGD. Top part: we used $\alpha \in \{1, 0.1\}$, $r \in \{1\%, 5\%, 10\%\}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\%, 10\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes. (No Serial SGD experiments due to memory limitations of our cluster.)

A.2 *Extended Numerical Results - Fault-Tolerant Setting*

If we run a distributed algorithm, for example on a shared computer cluster, then we may experience delays. Such delays can be caused by other processes running on the same compute node, node failures and/or for other reasons. As a result, given a computational (time) budget, these delays may cause nodes to fail to return a value. To illustrate this behavior, and to motivate the robust fault-tolerant L-BFGS method, we run a simple benchmark MPI code on two different environments:

- **Amazon EC2** – Amazon EC2 is a cloud system provided by Amazon. It is expected that if load balancing is done properly, the execution time will have small noise; however, the network and communication can still be an issue. (4 MPI processes)
- **Shared Cluster** – On our shared cluster, multiple jobs run on each node, with some jobs being more demanding than others. Even though each node has 16 cores, the amount of resources each job can utilize changes over time. In terms of communication, we have a GigaBit network. (11 MPI processes, running on 11 nodes)

We run a simple code on the cloud/cluster, with MPI communication. We generate two matrices $A, B \in R^{n \times n}$, then synchronize all MPI processes and compute $C = A \cdot B$ using the GSL C-BLAS library. The time is measured and recorded as computational time. After the matrix product is computed, the result is sent to the master/root node using asynchronous communication, and the time required is recorded. The process is repeated 3000 times.

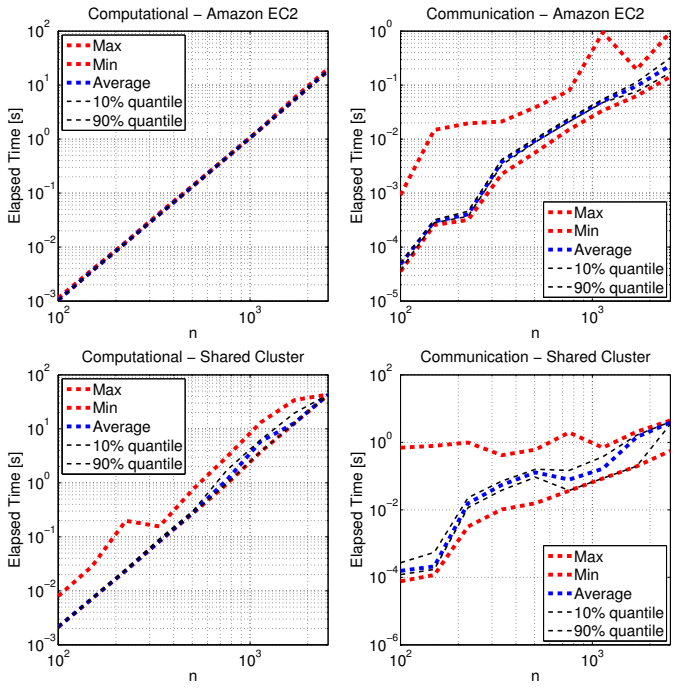


Figure A10. Distribution of Computation and Communication Time for Amazon EC2 and Shared Cluster. Figures show worst and best time, average time and 10% and 90% quantiles. Amazon Cloud EC: 4 MPI processes; Shared Cluster: 11 MPI processes.

The results of the experiment described above are captured in Figure A10. As expected, on the Amazon EC2 cloud, the matrix-matrix multiplication takes roughly the same time for all replications and the noise in communication is relatively small. In this example

the cost of communication is negligible when compared to the cost of computation. On our shared cluster, one cannot guarantee that all resources are exclusively used for a specific process, and thus, the computation and communication time is considerably more stochastic and unbalanced. In some cases, the difference between the minimum and maximum computation and communication times vary by an order of magnitude or more. Hence, on such a platform a fault-tolerant algorithm that only uses information from nodes that return an update within a preallocated budget is a natural choice.

In Figures A11-A14 we present a comparison of the proposed robust multi-batch L-BFGS method and the multi-batch L-BFGS method that does not enforce sample consistency (L-BFGS). In these experiments, p denotes the probability that a single node (MPI process) will not return a gradient evaluated on local data within a given time budget. We illustrate the performance of the methods for $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We observe that the robust implementation is not affected much by the failure probability p.

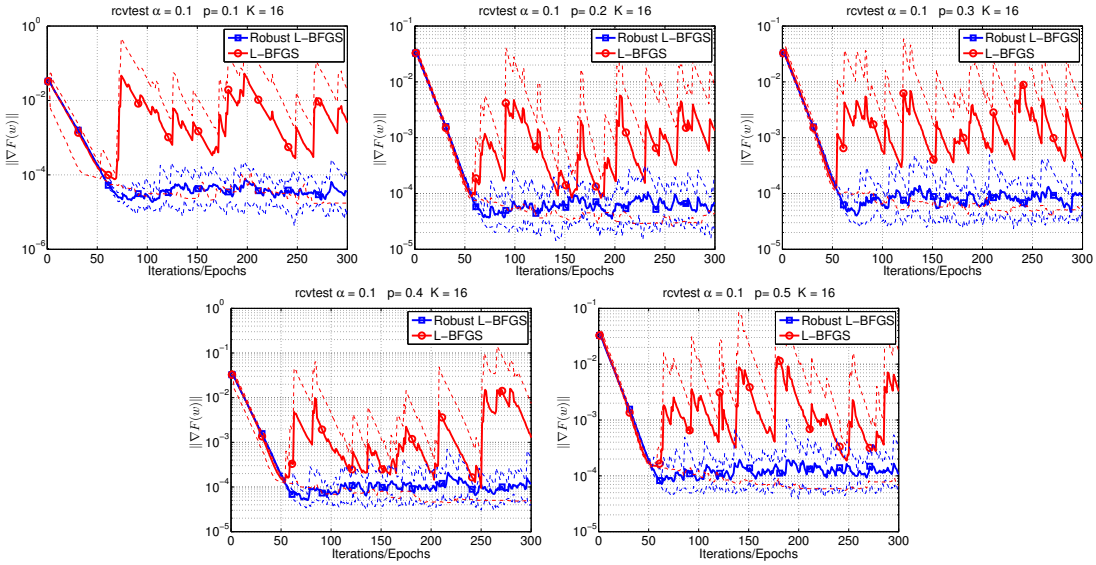


Figure A11. **rcvtest dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

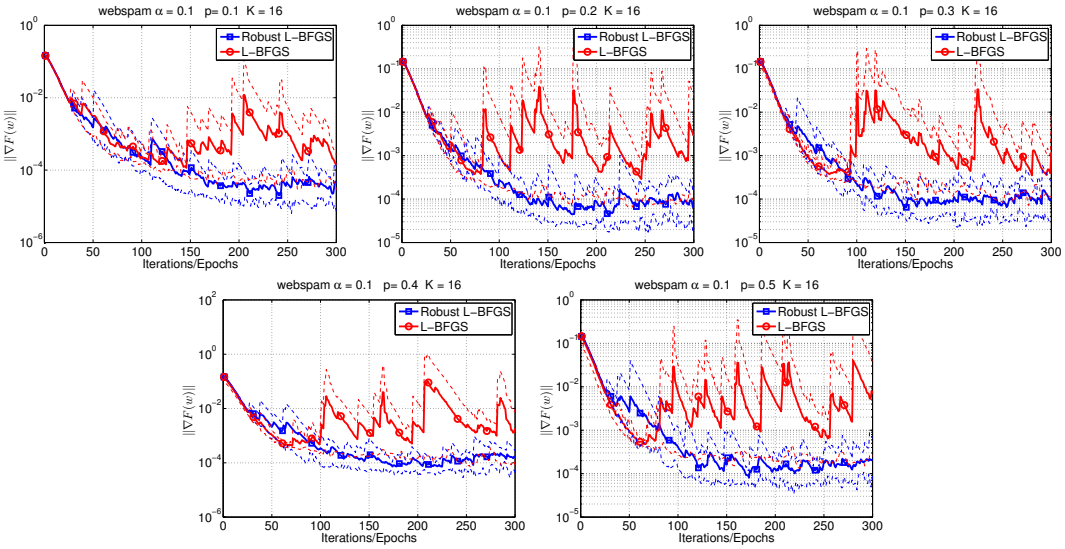


Figure A12. **webspam dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

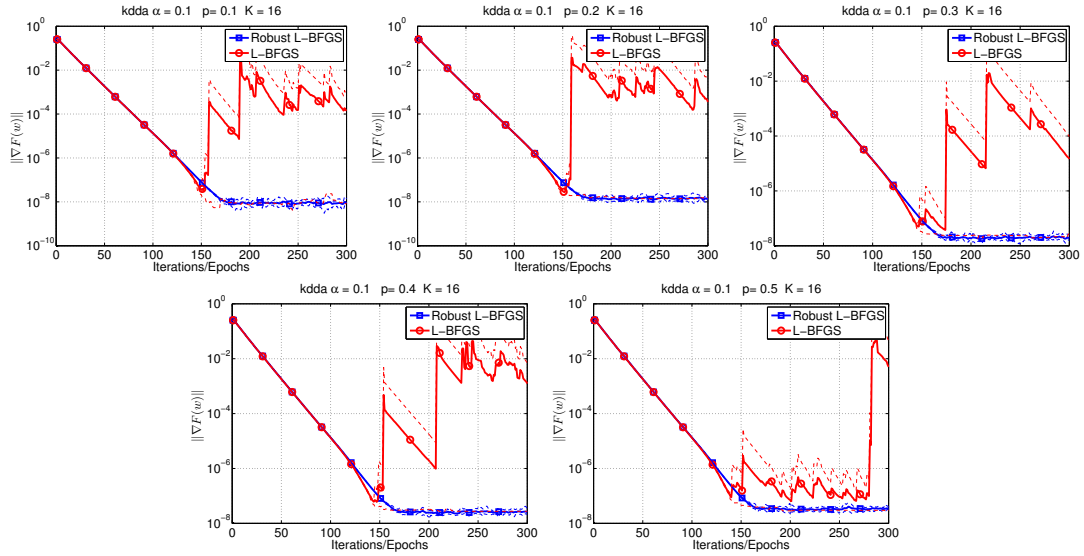


Figure A13. **kdda dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

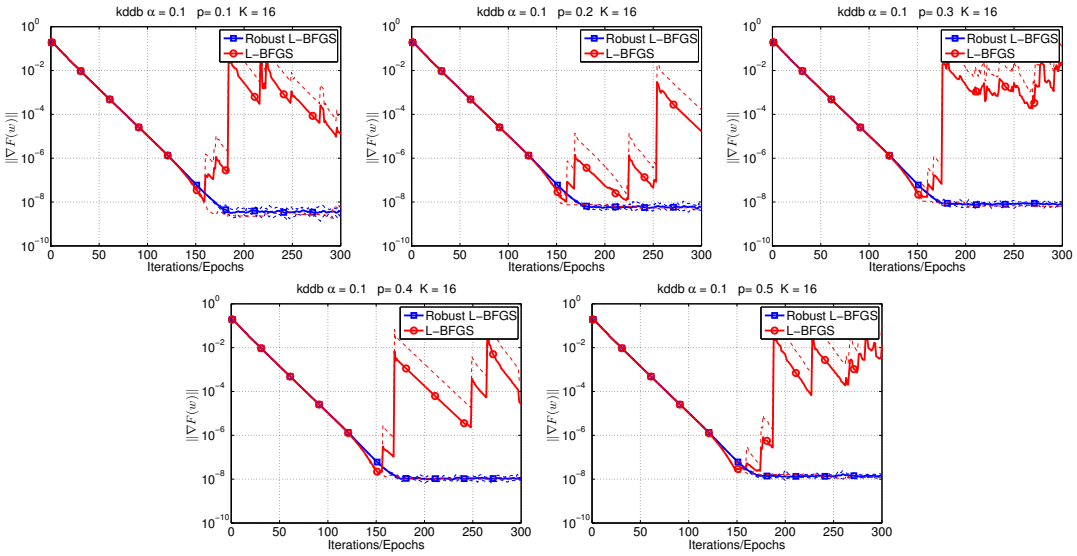


Figure A14. **kddb dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

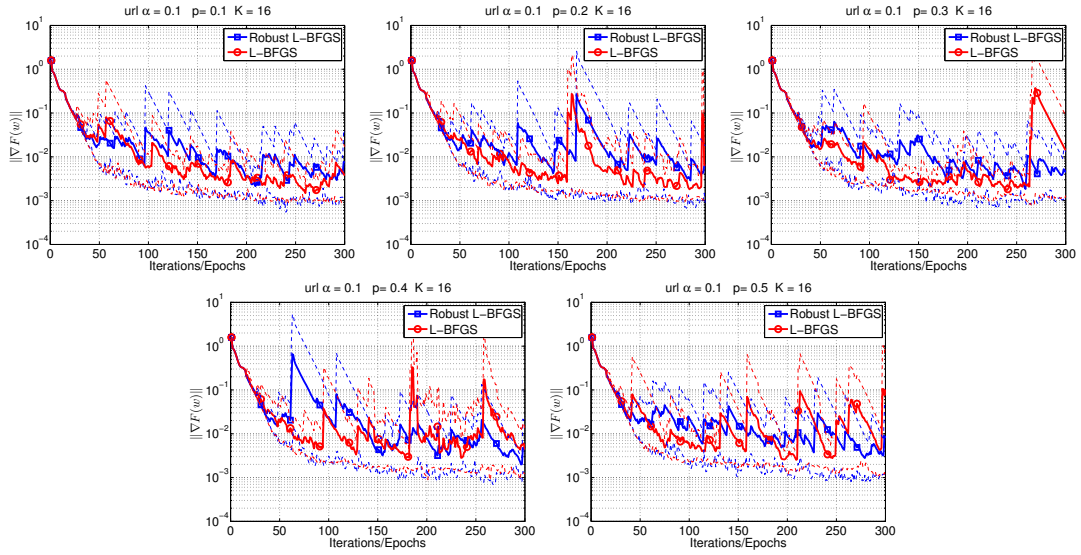


Figure A15. **url dataset**. Comparison of Robust L-BFGS and L-BFGS in the presence of faults. We used $\alpha = 0.1$ and $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

Appendix B. Extended Numerical Results - Neural Networks

In this section, we present complete numerical results for the Neural Network training tasks on the MNIST[5] and CIFAR10/CIFAR100[6] datasets. More specifically, we show the training and testing accuracy of the methods for all batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$). The tasks are summarized in Table B1, and the details are explained below.

Table B1. Structure of Neural Networks.

Network	Type	# of layers	d	Ref.
MNIST MLC	fully connected (FC)	1	7.8k	[39]
MNIST DNN (SoftPlus)	conv+FC	4	1.1M	[1]
MNIST DNN (ReLU)	conv+FC	4	1.1M	[1]
CIFAR10 LeNet	conv+FC	5	62.0k	[39]
CIFAR10 VGG11	conv+batchNorm+FC	29	9.2M	[64]
CIFAR100 VGG11	conv+batchNorm+FC	29	9.2M	[64]

MLC = Multiclass Linear Classifier

- **MNIST Convex:** This problem was a convex problem. It is a simple neural network with no hidden layers and soft-max cross-entropy loss function.
- **MNIST DNN (SoftPlus):** This network consists of two convolutional layers and two fully connected layer. The first convolutional layer has 32 filters for each 5×5 patch, and the second convolutional layer has 64 filters for each 5×5 patch. Every convolutional layer is followed by a 2×2 max pooling layer. The first fully connected layer has 1024 neurons and the second fully connected layer produces a 10 dimensional output. We used SoftPlus activation functions. The loss function is soft-max cross-entropy loss.
- **MNIST DNN (ReLU):** Same as above, but with ReLU activation functions.
- **CIFAR10 LeNet:** This network consists of two convolutional layers and three fully connected layer. The first convolutional layer has 6 filters for each 5×5 patch, and the second convolutional layer has 16 filters for each 5×5 patch. Every convolutional layer is followed by a 2×2 max pooling layer. The first fully connected layer has 120 neurons and the second fully connected layer has 84 neurons. The last fully connected layer produces a 10 dimensional output. The activation functions used is ReLU and the loss function used is soft-max cross-entropy loss.
- **CIFAR10 VGG11 and CIFAR100 VGG11:** This is standard VGG11 network which contains 8 convolution layers, each followed by batch-normalization and ReLU activation functions. There are also 5 max-pooling layers and one average pooling layer. The output of desired dimension (10 or 100) is achieve by fully connected layer. The loss function is soft-max cross-entropy loss.

For the experiments in this section (Figures B1 and B8), we ran the following methods:

- (LFBGS) multi-batch L-BFGS with the cautious strategy: Algorithm 1 with standard initial scaling ($\gamma_k I$, where $\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$) or αI initial scaling,
- (Adam) [35],
- (SGD) [60].

[5] Available at: http://yann.lecun.com/exdb/mnist/.

[6] Available at: https://www.cs.toronto.edu/~kriz/cifar.html.

Figures B1 and B6 show the evolution of training and testing accuracy for different batch sizes for the different neural network training tasks. Figures B7 and B8 show the two diagnostic metrics for different batch sizes.

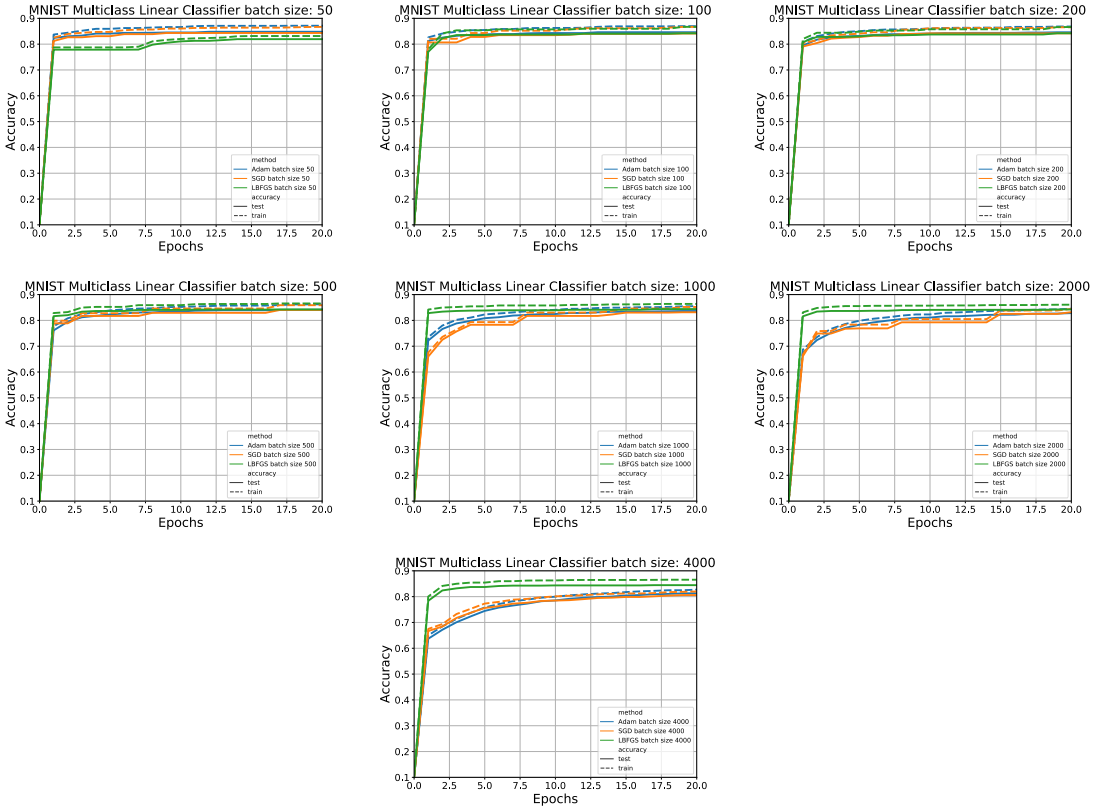


Figure B1. **MNIST Multiclass Linear Classifier.** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

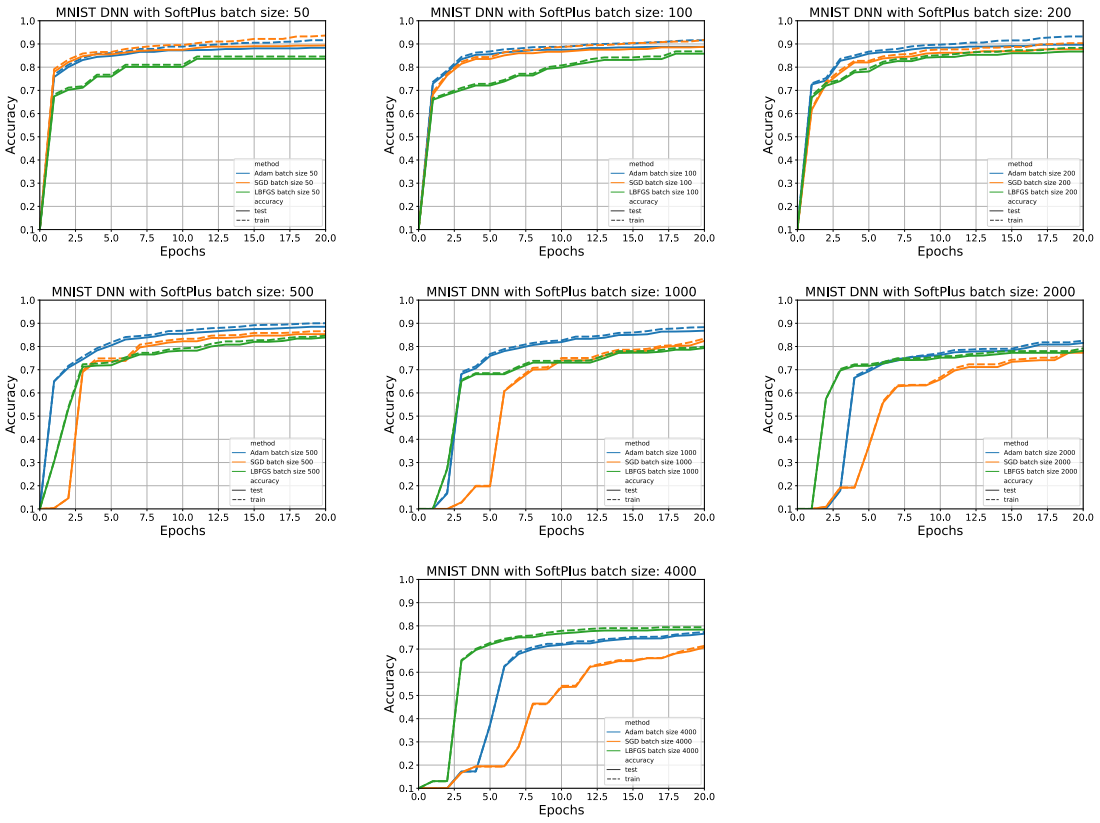


Figure B2. **MNIST DNN (SoftPlus).** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

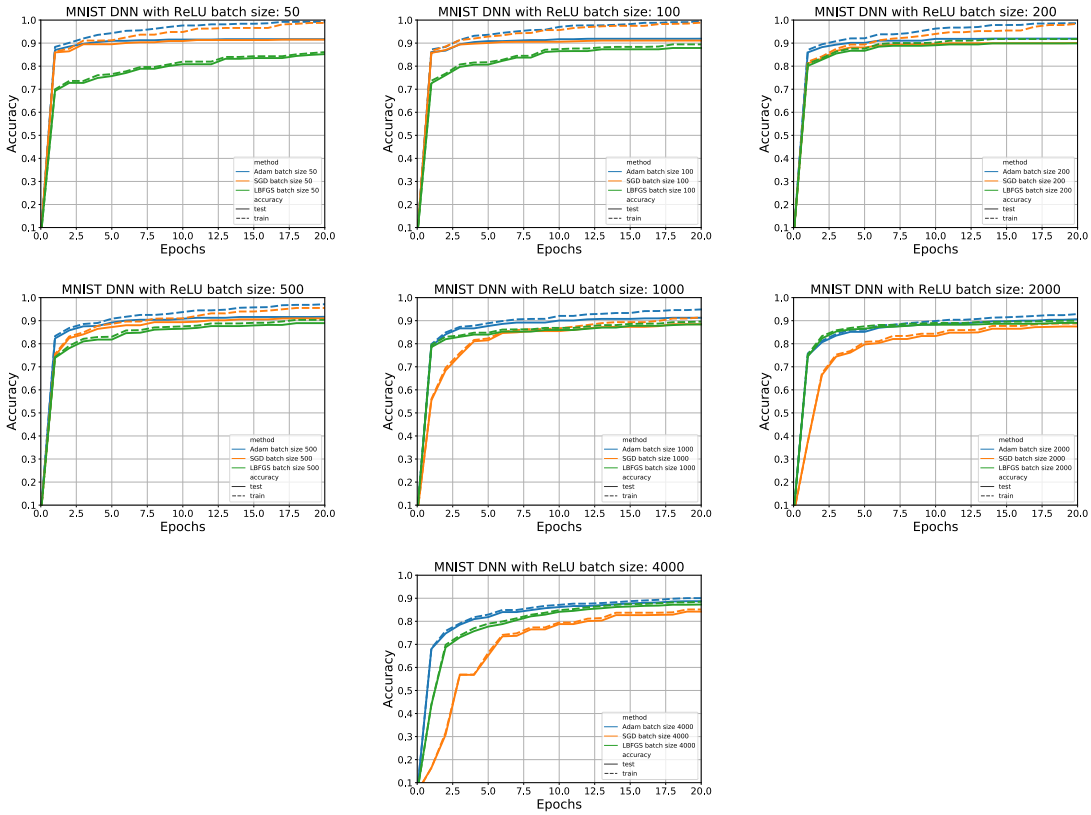


Figure B3. **MNIST DNN (ReLU).** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

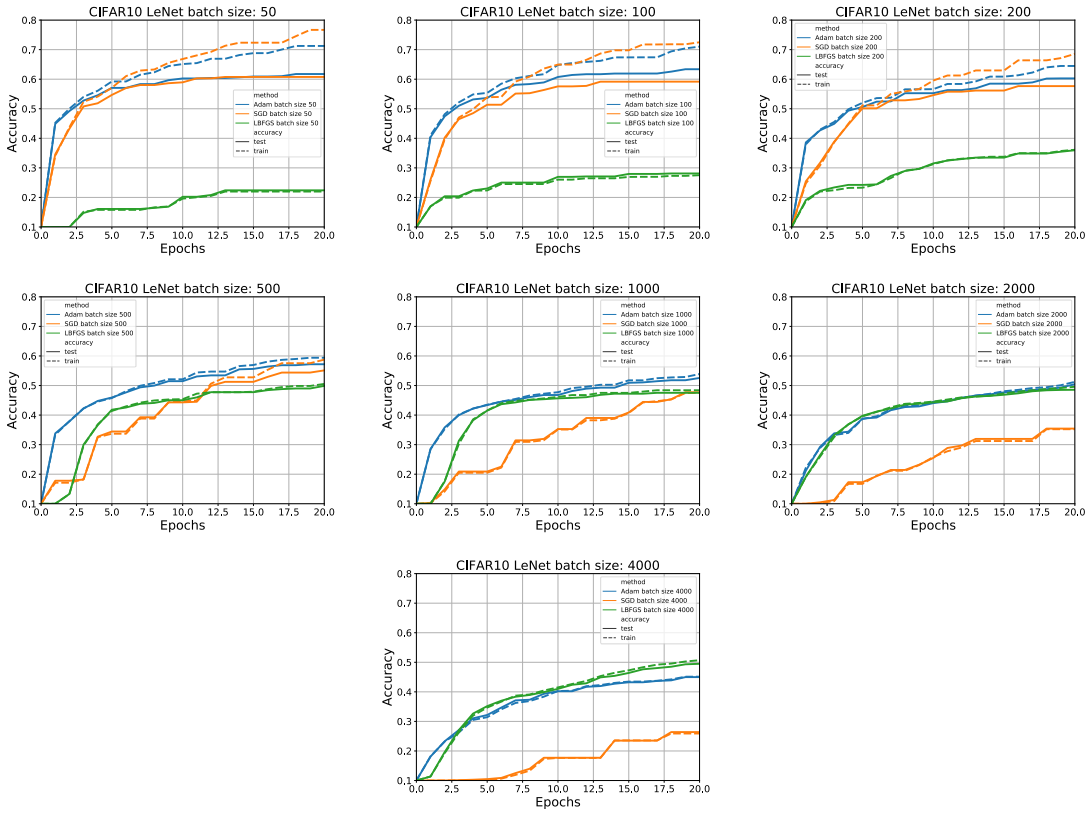


Figure B4. **CIFAR10 LeNet.** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

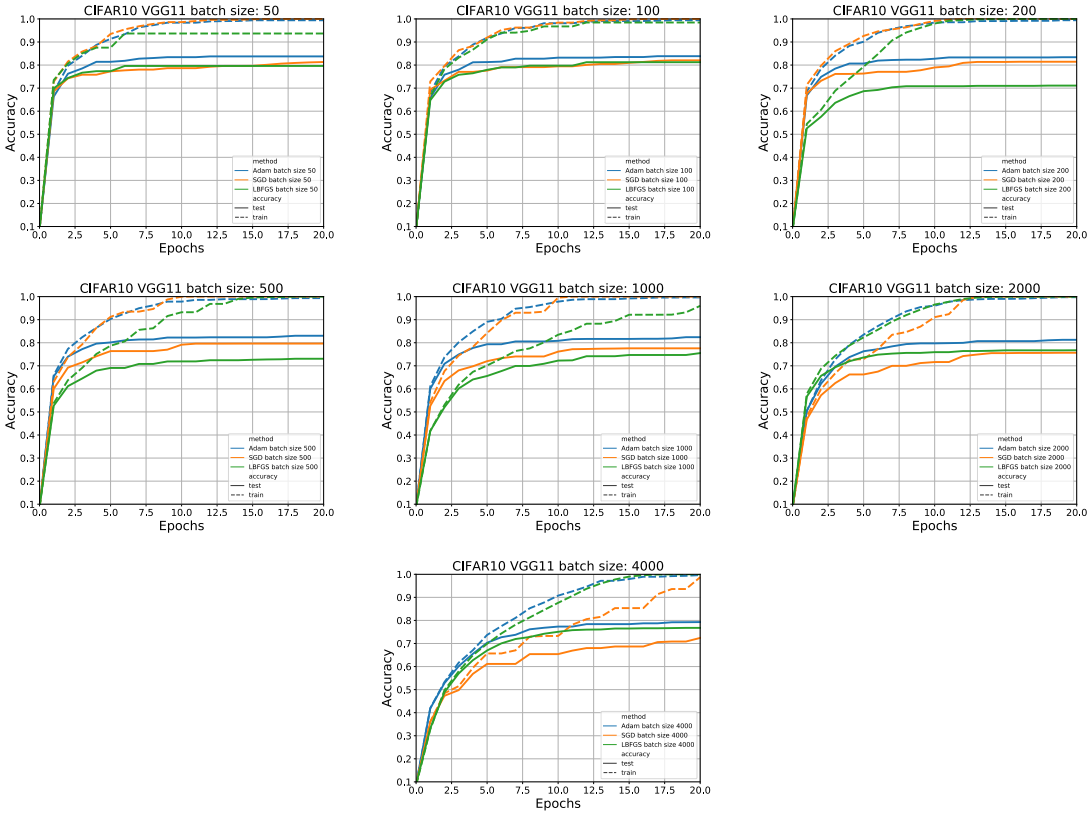


Figure B5. **CIFAR10 VGG11.** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

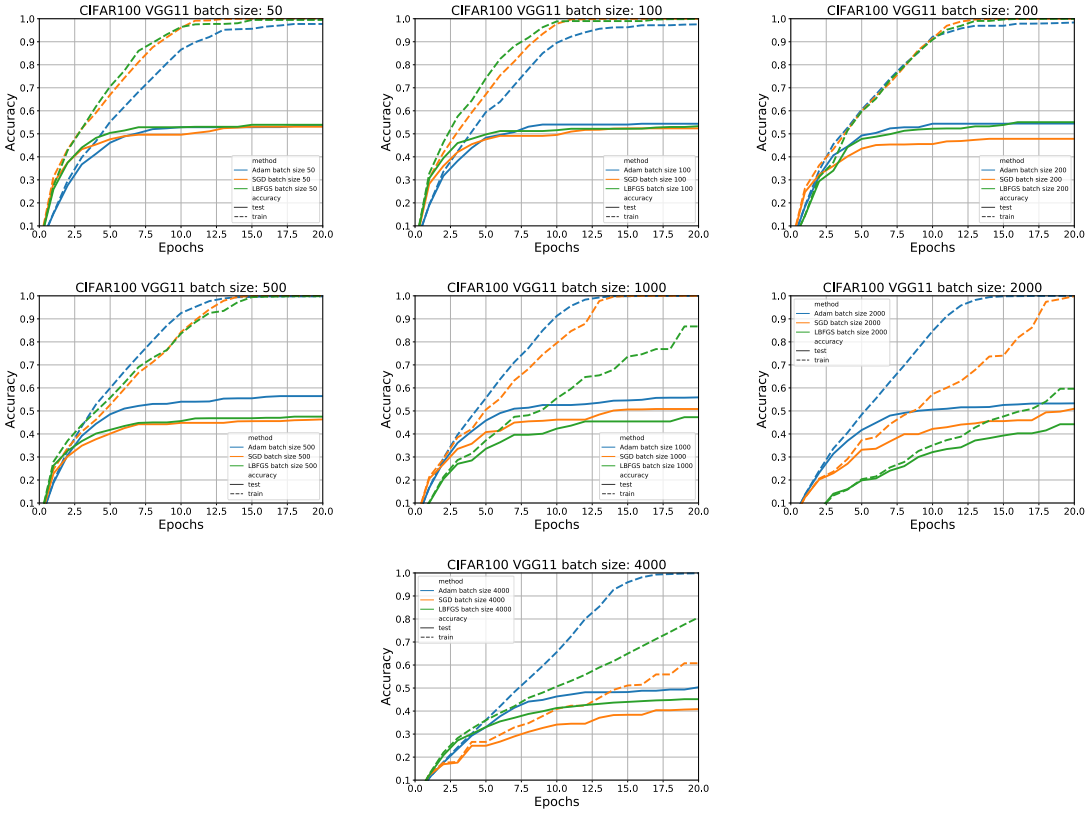


Figure B6. **CIFAR100 VGG11.** Comparison of multi-batch L-BFGS (LBFGS), Adam and SGD. Evolution of the training and testing accuracy for different batch sizes ($|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$).

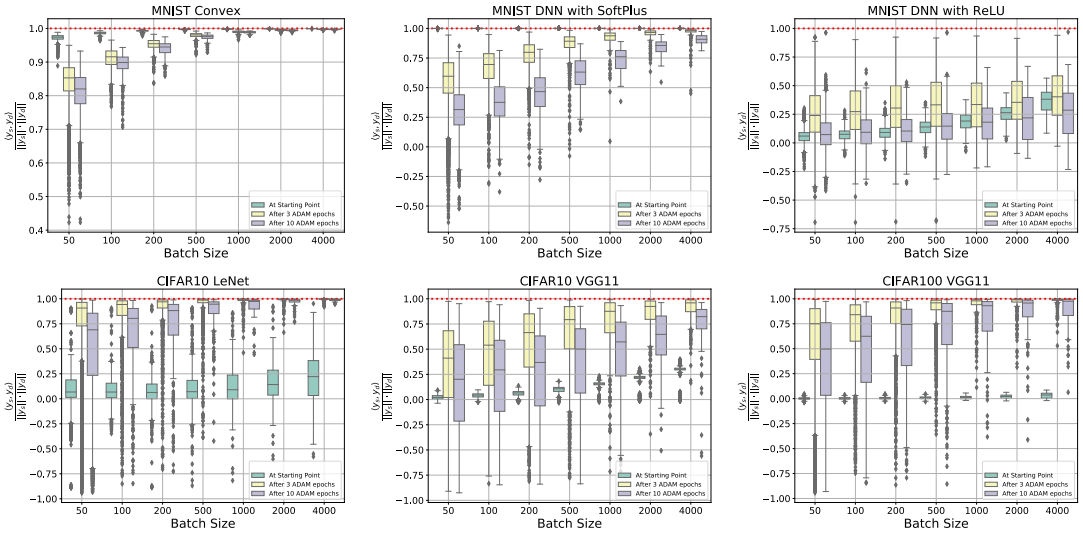


Figure B7. Multi-batch L-BFGS diagnostic metric for different batch sizes $|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$. Angle between the true gradient curvature vector y_d and subsampled gradient curvature vector y_s ($\frac{\langle y_s, y_d \rangle}{\|y_s\| \|y_d\|}$). Top right: MNIST Multiclass Linear Classifier; Top middle: MNIST DNN (SoftPlus); Top Right: MNIST DNN (ReLU); Bottom left: CIFAR10 LeNet; Bottom middle: CIFAR10 VGG11; Bottom right: CIFAR100 VGG11.

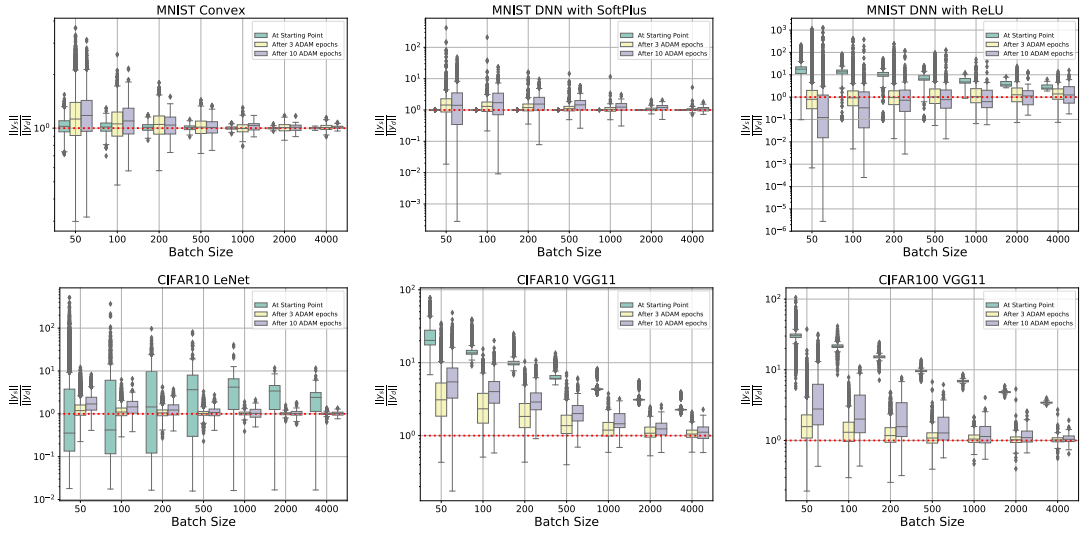


Figure B8. Multi-batch L-BFGS diagnostic metric for different batch sizes $|S| \in \{50, 100, 200, 500, 1000, 2000, 4000\}$. Ratio of subsampled gradient curvature vector to true gradient curvature vector ($\frac{y_s}{y_d}$). Top right: MNIST Multiclass Linear Classifier; Top middle: MNIST DNN (SoftPlus); Top Right: MNIST DNN (ReLU); Bottom left: CIFAR10 LeNet; Bottom middle: CIFAR10 VGG11; Bottom right: CIFAR100 VGG11.