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

Albert S. Berahas
Department of Engineering Sciences and Applied Mathematics
Northwestern University
Evanston, IL 60208, USA

Martin Takáč
Department of Industrial and Systems Engineering
Lehigh University
Bethlehem, PA 18015, USA

Abstract

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, consistency, machine learning.

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., (Tsitsiklis et al., 1986; Recht et al., 2011; Agarwal et al., 2014; Dean et al., 2012; Zinkevich et al., 2010; Das et al., 2016). In this paper we study a batch approach based on the L-BFGS method (Nocedal, 1980; Liu and Nocedal, 1989) 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 (Robbins and Monro, 1951; Bottou and Cun, 2003), and its variance-reduced and accelerated vari-

*. This work substantially extends the work of (Berahas et al., 2016) published at the Neural Information Processing Systems conference in 2016.
ants (Schmidt et al., 2016; Johnson and Zhang, 2013; Defazio et al., 2014; Nguyen et al., 2017a; Konečný et al., 2016; Konečný and Richtárik, 2017; He and Takáč, 2015; Nguyen et al., 2017b; Lin et al., 2015). 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 (Bertsekas and Tsitsiklis, 1989; Recht et al., 2011; Goodfellow et al., 2016; Reddi et al., 2015; Leblond et al., 2017; Takáč et al., 2013). 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 (Recht et al., 2011; Mania et al., 2015). As a result, the algorithm spends more time in communication as compared to computation. 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 (Takáč et al., 2013).

An alternative to SG-type methods are batch methods, such as L-BFGS (Nocedal, 1980), 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 (Zhang and Lin, 2015); however, batch methods are not as efficient learning algorithms as SG methods in a sequential setting (Bousquet and Bottou, 2007; Hardt et al., 2016). To benefit from both types of methods, some high performance machine learning systems implement both types of methods (Agarwal et al., 2014; Dean et al., 2012), and algorithms that transition from the stochastic to the batch regime (Friedlander and Schmidt, 2012; Byrd et al., 2012) 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. We call this a multi-batch approach 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. 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. The multi-batch approach can, however, cause difficulties to quasi-Newton methods because these methods employ gradient differences to update the Hessian approximations.

In this paper, we study how to design a robust multi-batch implementation of the limited-memory version of the classical BFGS method (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970)—which we call the multi-batch L-BFGS method—in the presence of two adverse situations (Berahas et al., 2016). 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 syn-
Multi-Batch L-BFGS

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., (Schraudolph et al., 2007; Bordes et al., 2009; Byrd et al., 2016; Curtis, 2016; Wang et al., 2015; Mokhtari and Ribeiro, 2015; Keskar and Berahas, 2016; Gower et al., 2016). The methods enumerated above differ in three major aspects: (i) the update rules for the curvature 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 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

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

where \((x, y)\) are input-output pairs, \(f: \mathbb{R}^d \rightarrow \mathbb{R}\) is the composition of a prediction function (parametrized by \(w\)) and a loss function, and \(\Omega\) 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

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

In a pure batch approach, one applies a gradient based method to this deterministic optimization problem (2.2). A popular method in this setting is the L-BFGS method (Liu and Nocedal, 1989). When \(n\) is large, it is natural to parallelize the evaluation of \(F\) and \(\nabla F\) by assigning the evaluation of the component functions \(f_i\) to different processors. If this is done in a distributed 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 be ignored given the stochastic nature of the true objective function (2.1). This leads, however, 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. Thus, we seek to find 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 the entire training set, \(T = \{(x^i, y^i)\},\) for \(i = 1, \ldots, n\), is not employed at every iteration but, rather, a subset of the data is used to compute the gradient. Specifically, we consider a method in which the dataset is randomly divided into a number of batches—say 10, 50, or 100—and the minimization is performed with respect to a different batch at each iteration. At the \(k\)-th iteration the algorithm chooses \(S_k \subset \{1, \ldots, n\}\), computes

$$F^{S_k}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} f_i(w_k), \quad g^{S_k}_k = \nabla F^{S_k}(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(w_k), \quad (2.3)$$

and takes a step along the direction \(-H_k g^{S_k}_k\), 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 of implementation 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 method are similar. The main difference is that node failures create unpredictable changes to the samples \(S_k\), whereas a multi-batch method has control over 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 and noisy gradient approximations. Our algorithm operates with much larger batches so that distributing the function and gradient evaluations is beneficial and the compute time of \(g^{S_k}_k\) 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 this paper 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

$$y_{k+1} = y_k^{O_k} - g_k^{O_k}, \quad s_{k+1} = w_{k+1} - w_k,$$  \hspace{1cm} (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 $F$ along the most recent displacement, and will lead 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 (Byrd et al., 2011; Bollapragada et al., 2016; Martens, 2010). 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 working nodes in a fault-tolerant parallel implementation, or by exerting a small degree of control over the creation of the samples $S_k$ in the multi-batch method, one can design a robust method that naturally builds upon the fundamental properties of BFGS updating.

We should mention in passing that a commonly used fix for ensuring stability of quasi-Newton updating in machine learning is to enforce gradient consistency (Schraudolph et al., 2007; Mokhtari and Ribeiro, 2015), 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 (Ngiam et al., 2011), 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 method), 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 $S_k \subset \{1, \ldots, n\}$ and computes a new iterate by the formula

$$w_{k+1} = w_k - \alpha_k H_k g_k^{S_k},$$  \hspace{1cm} (2.5)

where $\alpha_k$ 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

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k}, \quad V_k = 1 - \rho_k y_k s_k^T.$$  \hspace{1cm} (2.6)
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

\[
F^{O_k}(w_k) = \frac{1}{|O_k|} \sum_{i \in O_k} f_i(w_k), \quad g_{k}^{O_k} = \nabla F^{O_k}(w_k) = \frac{1}{|O_k|} \sum_{i \in O_k} \nabla f_i(w_k),
\]

and compute the correction vectors as in (2.4). This completely specifies the algorithm, except for the choice of step length \(\alpha_k\); 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 matrix \(H_k\) since one can obtain this product via the two-loop recursion (Nocedal and Wright, 1999), using the \(m\) most recent correction pairs \(\{s_i, y_i\}\). Employing this mechanism, the search direction can be computed in \(O(d)\) flops, 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 pseudo-code of the proposed 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 (MB-LBFGS)**

**Input:** \(w_0\) (initial iterate), \(T = \{(x^i, y^i), \text{ for } i = 1, \ldots, n\}\) (training set), \(m\) (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 the 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 and 6 in 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,
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 for the multi-batch setting: (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 1b 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 pass (epoch) all samples are used. This strategy has the advantage that it requires no extra computation in the evaluation of $g_{O_k}^k$ and $g_{O_k}^{k+1}$, 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 $\{1, \ldots, n\}$. The overlapping set $O_k$ is then formed by randomly selecting $|O_k|$ elements from $S_k$ (subsampling). This strategy is slightly more expensive since $g_{O_k}^{k+1}$ 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. We assume that the goal is to minimize the empirical risk $F$ given in (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 $w^*$, but only convergence to a neighborhood of $w^*$ (Nedić and Bertsekas, 2001). 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 is a constant $\gamma$ such that $\mathbb{E}_S[\|\nabla F^S(w)\|^2] \leq \gamma^2$ 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., $\mathbb{E}_S[\nabla F^S(w)] = \nabla F(w)$. 

8
MULTI-BATCH L-BFGS

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

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

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 (Byrd et al., 2016).

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

$$\mu_1 I \preceq H_k \preceq \mu_2 I, \quad \forall k = 0, 1, 2, \ldots$$

**Proof** Instead of analyzing the inverse Hessian approximation $H_k$, we study the direct 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 $i = 0, \ldots, \tilde{m} - 1$ set $j = k - \tilde{m} + 1 + i$ and compute
   $$B_k^{(i+1)} = B_k^{(i)} - \frac{B_k^{(i)} s_j^T B_k^{(i)} s_j}{s_j^T B_k^{(i)} s_j} + \frac{y_j y_j^T}{y_j^T s_j}.$$ 
3. Set $B_{k+1} = B_k^{(\tilde{m})}$.

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

$$y_{k+1} = y_{k+1}^O - y_k^O, \quad s_{k+1} = w_{k+1} - w_k.$$  

(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

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

(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,

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

(3.4)

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

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

(3.5)
The above proves that the eigenvalues of the matrices \( B^{(0)}_k = \frac{y_i^T s_{ji}^k}{s_{ji}^k} I \) 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,

\[
tr(B_{k+1}) = tr(B^{(0)}_k) - tr \left( \sum_{i=1}^{\tilde{m}} \frac{B^{(i-1)}_k s_{ji}^k T B^{(i-1)}_k s_{ji}^k}{s_{ji}^k B^{(i-1)}_k s_{ji}^k} \right) + tr \left( \sum_{i=1}^{\tilde{m}} \frac{y_i y_{ji}^T}{y_{ji}^T s_{ji}} \right)
\]

\[
\leq tr(B^{(0)}_k) + \sum_{i=1}^{\tilde{m}} \| y_{ji} \|^2 \leq tr(B^{(0)}_k) + \tilde{m} \hat{\lambda} \leq C_1,
\]

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 \( B^{(0)}_k \) are bounded above and away from zero.

Using a result due to Powell (1976), the determinant of the matrix \( B_{k+1} \) generated by the multi-batch L-BFGS method can be expressed as,

\[
\det(B_{k+1}) = \det(B^{(0)}_k) \prod_{i=1}^{\tilde{m}} \frac{y_{ji}^T s_{ji}^k}{s_{ji}^k B^{(i-1)}_k s_{ji}^k} = \det(B^{(0)}_k) \prod_{i=1}^{\tilde{m}} \frac{y_{ji}^T s_{ji}^k}{s_{ji}^k B^{(i-1)}_k s_{ji}^k}
\]

\[
\geq \det(B^{(0)}_k) \left( \frac{\hat{\lambda}}{C_1} \right)^{\tilde{m}} \geq C_2,
\]

for some positive constant \( C_2 \), where the above inequalities are due to the fact that the largest eigenvalue of \( B^{(i)}_k \) 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 \( F^* = F(w^*) \), where \( w^* \) is the minimizer of \( F \). Then, for all \( w \in \mathbb{R}^d \),

\[
2\lambda(F(w) - F^*) \leq \| \nabla F(w) \|^2.
\]

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

\[
F(x) \leq F(y) + \nabla F(y)^T (x - y) + \frac{1}{2\lambda} \| \nabla F(y) - \nabla F(x) \|^2.
\]

1. (Nesterov, 2013, Chapter 2.1.3)
Let $x = w$ and $y = w^*$

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

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

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 $F^* = F(w^*)$, where $w^*$ is the minimizer of $F$. Let $\{w_k\}$ be the iterates generated by Algorithm 1 with

$$\alpha_k = \alpha \in (0, \frac{1}{2\mu_1\lambda}),$$

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

$$\mathbb{E}[F(w_k) - F^*] \leq (1 - 2\alpha\mu_1\lambda)^k [F(w_0) - F^*] + [1 - (1 - 2\alpha\mu_1\lambda)^k] \frac{\alpha\mu_2^2\gamma^2\Lambda}{4\mu_1\lambda}$$

$$\xrightarrow{k \to \infty} \frac{\alpha\mu_2^2\gamma^2\Lambda}{4\mu_1\lambda}.$$  

**Proof** We have that

$$F(w_{k+1}) = F(w_k - \alpha H_k \nabla F^{S_k}(w_k))$$

$$\leq F(w_k) + \nabla F(w_k)^T (-\alpha H_k \nabla F^{S_k}(w_k)) + \frac{\Lambda}{2} \|\alpha H_k \nabla F^{S_k}(w_k)\|^2$$

$$\leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F^{S_k}(w_k) + \frac{\alpha^2 \mu_2^2 \Lambda}{2} \|\nabla F^{S_k}(w_k)\|^2,$$  

(3.8)

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 (3.8)

$$\mathbb{E}_{S_k}[F(w_{k+1})] \leq F(w_k) - \alpha \nabla F(w_k)^T H_k \nabla F(w_k) + \frac{\alpha^2 \mu_2^2 \Lambda}{2} \mathbb{E}_{S_k}[\|\nabla F^{S_k}(w_k)\|^2]$$  

(3.9)

$$\leq F(w_k) - \alpha \mu_1 \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2},$$  

(3.10)

where in the first inequality we make use of Assumption A.4, and the second inequality arises due to Lemma 3.1 and Assumption A.3. Since $F$ is $\lambda$-strongly convex, we can substitute the result of Lemma 3.2 in (3.10),

$$\mathbb{E}_{S_k}[F(w_{k+1})] \leq F(w_k) - \alpha \mu_1 \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2}$$

$$\leq F(w_k) - 2\alpha \mu_1 \lambda [F(w_k) - F^*] + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2}.$$  

(3.11)
Let
\[ \phi_k = \mathbb{E}[F(w_k) - F^*], \] (3.12)
where the expectation is over all batches \( S_0, S_1, ..., S_{k-1} \) and all history starting with \( w_0 \).

Equation (3.11) can be expressed as,
\[ \phi_{k+1} \leq (1 - 2\alpha \mu_1 \lambda)\phi_k + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2}, \] (3.13)
from which we deduce that in order to reduce the function value with respect to the previous function value, the step length needs to be in the range
\[ \alpha \in \left( 0, \frac{1}{2\mu_1 \lambda} \right). \]

Subtracting \( \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} \) from either side of (3.13) yields
\[
\phi_{k+1} - \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} \leq (1 - 2\alpha \mu_1 \lambda)\phi_k + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2} - \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} = (1 - 2\alpha \mu_1 \lambda) \left[ \phi_k - \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} \right]. \] (3.14)

Recursive application of (3.14) yields
\[
\phi_k - \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} \leq (1 - 2\alpha \mu_1 \lambda)^k \left[ \phi_0 - \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda} \right],
\]
and thus,
\[
\phi_k \leq (1 - 2\alpha \mu_1 \lambda)^k \phi_0 + \left[ 1 - (1 - 2\alpha \mu_1 \lambda)^k \right] \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda}.
\]

Finally using the definition of \( \phi_k \) (3.12) with the above expression yields the desired result
\[ \mathbb{E}[F(w_k) - F^*] \leq (1 - 2\alpha \mu_1 \lambda)^k [F(w_0) - F^*] + \left[ 1 - (1 - 2\alpha \mu_1 \lambda)^k \right] \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{4\mu_1 \lambda}. \]

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 that 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 will consider again these tradeoffs in Section 4, where we also note that larger batches 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 version of the algorithm \( H_k = I \), the step length range prescribed by the multi-batch L-BFGS method

\[ \]
depends on $\mu_1$, the smallest eigenvalue 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 worse than that of the first-order variant if the update has been unfortunate and generates a very small eigenvalue $\mu_1$. 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 $w^*$ by employing a sequence of step lengths $\{\alpha_k\}$ that converge to zero according to the schedule proposed by Robbins and Monro (1951). 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 (Byrd et al., 2016, Theorem 3.2) for more details and the proof.

**Theorem 3.4** Suppose that Assumptions A.1-A.4 hold, and let $F^* = F(w^*)$, where $w^*$ is the minimizer of $F$. Let $\{w_k\}$ be the iterates generated by Algorithm 1 with

$$\alpha_k = \frac{\beta}{k+1} \quad \text{and} \quad \beta > \frac{1}{2\mu_1\lambda},$$

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

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

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

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 (Schmidt et al., 2016; Johnson and Zhang, 2013; Defazio et al., 2014; Nguyen et al., 2017a; Konečný et al., 2016; Nguyen et al., 2017b; Lin et al., 2015). 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 (Berahas and Takáč, 2017).

### 3.2 Nonconvex case

The BFGS method is known to fail on nonconvex problems (Mascarenhas, 2004; Dai, 2002). 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 BFGS method in the nonconvex setting several techniques have been proposed including cautious updating (Li and Fukushima, 2001a), modified updating (Li and Fukushima, 2001b) and damping (Powell, 1978). Here we employ a cautious strategy that is well suited to our particular algorithm; we skip the update, i.e., set $H_{k+1} = H_k$, if the curvature condition

$$y_k^T s_k \geq \epsilon \|s_k\|^2$$

(3.17)
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 $\Lambda$-Lipschitz continuous for all $w \in \mathbb{R}^d$; the gradients of $F^S$ are $\Lambda_S$-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 $\Lambda_O$-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 $\hat{F}$.

4. There exist constants $\gamma \geq 0$ and $\eta > 0$ such that $E_S[\|\nabla F^S(w)\|_2^2] \leq \gamma^2 + \eta \|\nabla F(w)\|_2$ 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., $E_S[\nabla F^S(w)] = \nabla F(w)$.

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 Hessian approximations generated by Algorithm 1, with the modification that the Hessian approximation update is performed only when (3.17) is satisfied, for some $\epsilon > 0$, else $H_{k+1} = H_k$. Then, there exist constants $0 < \mu_1 \leq \mu_2$ such that

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

**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

$$y_{k+1} = g^O_{k+1} - g^O_k, \quad s_{k+1} = w_{k+1} - w_k.$$ 

The skipping mechanism (3.17) provides both an upper and lower bound on the quantity $\frac{\|y_k\|^2}{y_k^Ts_k}$, 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 (3.17). We have from (3.17) that

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

from which it follows that

$$s_k^Ty_k \leq \|s_k\|\|y_k\| \leq \frac{1}{\epsilon}\|y_k\|^2 \quad \Rightarrow \quad \frac{\|y_k\|^2}{s_k^Ty_k} \geq \epsilon. \quad (3.18)$$
The upper bound is attained by the Lipschitz continuity of sample gradients (Assumption B.2),
\[ y_k^T s_k \geq \epsilon \| s_k \|^2 \geq \epsilon \frac{\| y_k \|^2}{\Lambda O} \Rightarrow \frac{\| y_k \|^2}{s_k^T y_k} \leq \frac{\Lambda^2}{\epsilon}. \]

Combining (3.18) and (3.19),
\[ \epsilon \leq \frac{\| y_k \|^2}{s_k^T y_k} \leq \frac{\Lambda^2}{\epsilon}. \]

The above proves that the eigenvalues of the matrices \( B_k^{(0)} = y_k^T y_k I \) 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 (Bottou et al., 2016, 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 Hessian approximation update is performed only when (3.17) is satisfied, for some \( \epsilon > 0 \), else \( H_{k+1} = H_k \), and where
\[ \alpha_k = \alpha \in (0, \frac{\mu_1}{\mu_2 \eta \Lambda}), \]
and \( w_0 \) is the starting point. Then, for all \( k \geq 0 \),
\[ \mathbb{E}[\frac{1}{\tau} \sum_{k=0}^{\tau-1} \| \nabla F(w_k) \|^2] \leq \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{\mu_1} + \frac{2[F(w_0) - \hat{F}]}{\alpha \mu_1 \tau} \]
\[ \xrightarrow{\tau \to \infty} \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{\mu_1}. \]

**Proof** Starting with (3.9) and taking an expectation over the batch \( S_k \),
\[ \mathbb{E}_{S_k}[F(w_{k+1})] \leq F(w_k) - \alpha \mu_1 \| \nabla F(w_k) \|^2 + \frac{\alpha^2 \mu_2^2 \Lambda}{2} \mathbb{E}_{S_k} \left[ \| \nabla F_{S_k}(w_k) \|^2 \right] \]
\[ \leq F(w_k) - \alpha \mu_1 \| \nabla F(w_k) \|^2 + \frac{\alpha^2 \mu_2^2 \Lambda}{2} \left( \gamma^2 + \eta \| \nabla F(w) \|^2 \right) \]
\[ = F(w_k) - \alpha \left( \mu_1 - \frac{\alpha \mu_2^2 \eta \Lambda}{2} \right) \| \nabla F(w_k) \|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2} \]
\[ \leq F(w_k) - \frac{\alpha \mu_1}{2} \| \nabla F(w_k) \|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2}, \]

(3.20)
where the second inequality holds due to Assumption B.4, and the fourth inequality is obtained by using the upper bound on the step length. Taking the expectation over all batches $S_0, S_1, \ldots, S_{k-1}$ and all history starting with $w_0$ yields

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

(3.21)

where $\phi_k = \mathbb{E}[F(w_k)]$. Summing (3.21) over the first $\tau$ iterations

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

(3.22)

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

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

Substituting the above expression into (3.22) and rearranging terms

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

Dividing the above equation by $\tau$ 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 (Ghadimi and Lan, 2013). 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 Hessian approximation update is performed only when (3.17) 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 $\tau > \frac{\epsilon^2 \mu_2^2 \eta^2 \Lambda^2}{4 \mu_1^2}$, and $w_0$ is the starting point. Then,

$$
\min_{0 \leq k \leq \tau - 1} \mathbb{E}[\|\nabla F(w_k)\|^2] \leq \frac{2(F(w_0) - \hat{F}) \mu_2^2 \gamma^2 \Lambda}{\delta(\alpha)^2 \tau}.
$$
Proof Starting with (3.20)

$$\mathbb{E}_{S_k}[F(w_{k+1})] \leq F(w_k) - \alpha (\mu_1 - \frac{\alpha \mu_2^2 \eta \Lambda}{2}) \|\nabla F(w_k)\|^2 + \frac{\alpha^2 \mu_2^2 \gamma^2 \Lambda}{2},$$

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 (3.23) yields

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

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

$$\min_{0 \leq k \leq \tau - 1} \mathbb{E}[\|\nabla F(w_k)\|^2] \leq \frac{1}{\tau} \sum_{k=0}^{\tau-1} \mathbb{E}[\|\nabla F(w_k)\|^2] \leq \frac{1}{\alpha \delta(\alpha) \tau} \mathbb{E}[F(w_0) - F(w_\tau)] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2 \delta(\alpha)}$$

$$\leq \frac{1}{\alpha \delta(\alpha) \tau} [F(w_0) - \hat{F}] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2 \delta(\alpha)}$$

$$\leq \frac{1}{\delta(\frac{c}{\sqrt{\tau}}) c \sqrt{\tau}} [F(w_0) - \hat{F}] + \frac{\alpha \mu_2^2 \gamma^2 \Lambda}{2 \delta(\frac{c}{\sqrt{\tau}}) \sqrt{\tau}}.$$

The first inequality holds because the minimum value is less than the average value, and the third inequality holds because $\hat{F} \leq F(x_\tau)$ (Assumption B.3). The last expression comes as a result of using the definition of the step length, $\alpha = \frac{c}{\sqrt{\tau}}$. Setting

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

yields the desired result.

We now comment on the quantity $\delta(\alpha)$ that first appears in (3.23), 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 (3.24), we require that

$$\alpha = \frac{c}{\sqrt{\tau}} < \frac{2 \mu_1}{\mu_2^2 \eta \Lambda}.$$  (3.25)

In order to ensure that (3.25) holds, we impose that

$$\tau > \frac{(F(w_0) - \hat{F}) \mu_2^2 \gamma^2 \Lambda}{2 \gamma^2 \mu_1^2} = \frac{c^2 \mu_2^2 \gamma^2 \Lambda^2}{4 \mu_1^2}.$$
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 $\tau$, the total number of iteration. In practice, one would use $\alpha_k = \frac{1}{\sqrt{k}}$, which would result in a $O\left(\frac{1}{\sqrt{k}}\right)$ 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,

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

where $(x^i, y^i)_{i=1}^{n}$ denote the training examples and $\sigma = \frac{1}{n}$ is the regularization parameter. We present numerical results that evaluate 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\textsuperscript{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.

\textsuperscript{2} LIBSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
multi-batch L-BFGS

Fig. 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 $o \in \{5\%, 10\%, 20\%, 30\\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.

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 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 on two Neural Network tasks, both on the MNIST dataset\(^3\). Table 1 summarizes the two network architectures that we used; see Appendix B for more details. The first network ($DNN$) is a two-layer Convolutional Neural Network with cross-entropy loss, and the second network ($Regression$) is a Feed-Forward Neural Network with no hidden units and $\ell_2$ loss function. The second problem is convex.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|l|}
\hline
Network & Type & Structure \\
\hline
$DNN$ & Convolutional & Conv(32 features, 5x5 patch) - Max pool (2x2) - Conv(64 features, 5x5 patch) - Max pool (2x2) - Fully (1024) Softmax cross entropy loss function \\
$Regression$ & Feed-forward & NN with no hidden layers $\ell_2$ loss function (convex problem) \\
\hline
\end{tabular}
\caption{Structure of Neural Networks.}
\end{table}

We implemented our algorithm in TensorFlow Abadi et al. (2015) and compare against several popular and readily available algorithms: (i) SGD, (ii) Adagrad (Duchi et al., 2011), (iii) Adadelta (Zeiler, 2012), and (iv) Adam (Kingma and Ba, 2014). For each method, we conducted a grid search to find the best learning rate $\alpha \in \{2^{-2}, \ldots, 2^{-10}\}$, and also investigated the effect of different batch sizes $|S| \in \{32, 64, \ldots, 4096\}$; see Appendix B for detailed experiments with all batch sizes. For the multi-batch L-BFGS method we also investigated the effect of history length $m \in \{5, 10, 20\}$. The overlap used in our proposed method was 25% of the batch, $o = 0.25$.

Keskar and Berahas (2016) observed that the widely used Barzilai-Borwein-type scaling $s_k^T y_k y_k^T I$ of the initial Hessian approximation may lead to quasi-Newton updates that are not
Multi-Batch L-BFGS

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 $\alpha I$.

Figure 4: DNN - MNIST. Comparison of the multi-batch L-BFGS variants (LBFGS, LBFGS2), Adam, Adagrad, Adadelta and SGD. Top part: Evolution of the training and testing accuracy for different batch sizes. Bottom part: Zoomed in version of the top part for high accuracy.

Figure 5: Regression - MNIST. Comparison of the multi-batch L-BFGS variants (LBFGS, LBFGS2), Adam, Adagrad, Adadelta and SGD. Top part: Evolution of the training and testing accuracy for different batch sizes. Bottom part: Zoomed in version of the top part for high accuracy.

Figures 4 and 5 depict the evolution of the best training (solid line) and testing accuracy (dashed line) achieved by the methods on the DNN and Regression tasks, respectively, over the first 20 epochs of training for various batch sizes. The second row of the figures are
zoomed in versions of the first rows, and clearly illustrate the evolution of the training and testing for the different methods in the high accuracy regime. For some experiments (e.g., DNN with batch size 64), there is a sudden drop in the performance of the LBFGS2 method (green line), this can be attributed to numerical instability, and more specifically float overflows that occur. Overall, one can observe that for small batch sizes, the multi-batch L-BFGS variants perform worse than the other methods on the DNN problem. However, when large batches are employed, LBFGS and LBFGS2 perform at least on par with the other method on the DNN task, and outperforms the other methods on the Regression task.

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 underutilization of computational hardware and in fact hinders the ability to parallelize the methods. Figure 6 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 TensorFlow. It is clear that for the Regression task, the compute time of a single gradient is roughly the same as the compute time of a gradient based on a batch size of 512, whereas for the DNN task, the compute time of the gradients appear constant up to batch sizes of roughly 128.

![Figure 6: Relative slow-down of computational time of for different batch-size to computational time with batch-size 1.](image)

### 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 7 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...
Multi-Batch L-BFGS

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 7: 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). +C indicates that we include communication time to the gradient computation and L-BFGS update computation.

**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 8. On the other hand, because we are using a Vector-Free L-BFGS implementation (Chen et al., 2014) 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.

**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 9 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.

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 the 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
Multi-Batch L-BFGS

maintaining useful curvature information. The proposed 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. More 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.

Acknowledgments

We would like to thank Professor Jorge Nocedal for his insightful comments and for helping us write the conference version of this paper. Albert S. Berahas was supported in part by Department of Energy grant DE-FG02-87ER25047 and Martin Takáč was supported by NSF Grants CCF-1618717 and CMMI-1663256.
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 A, 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>
<thead>
<tr>
<th>Dataset</th>
<th>n</th>
<th>d</th>
<th>Size (MB)</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>ijcnn (test)</td>
<td>91,701</td>
<td>22</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>cov</td>
<td>581,012</td>
<td>54</td>
<td>68</td>
<td>4</td>
</tr>
<tr>
<td>news20</td>
<td>19,996</td>
<td>1,355,191</td>
<td>134</td>
<td>4</td>
</tr>
<tr>
<td>rcvtest</td>
<td>677,399</td>
<td>47,236</td>
<td>1,152</td>
<td>16</td>
</tr>
<tr>
<td>url</td>
<td>2,396,130</td>
<td>3,231,961</td>
<td>2,108</td>
<td>16</td>
</tr>
<tr>
<td>kdda</td>
<td>8,407,752</td>
<td>20,216,830</td>
<td>2,546</td>
<td>16</td>
</tr>
<tr>
<td>kddb</td>
<td>19,264,097</td>
<td>29,890,095</td>
<td>4,894</td>
<td>16</td>
</tr>
<tr>
<td>webspam</td>
<td>350,000</td>
<td>16,609,143</td>
<td>23,866</td>
<td>16</td>
</tr>
<tr>
<td>splice-site</td>
<td>50,000,000</td>
<td>11,725,480</td>
<td>260,705</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 2: Datasets and basic statistics. All datasets are available at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

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

\[
\min_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{y_i(w^T x_i)}) + \frac{\sigma}{2} \|w\|^2,
\]

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 10-18), 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 10-18 we show the evolution of \(\|\nabla F(w)\|\) for different step lengths \(\alpha\), and for various batch (\(|S| = r \cdot n\)) and overlap (\(|O| = o \cdot |S|\)) sizes. Each Figure (10-18) 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 10: \textbf{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 $o \in \{5\%, 10\%, 20\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.
Figure 11: 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 $o \in \{5\%, 10\%, 20\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.
Figure 12: **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 $o \in \{5\%, 10\%, 20\%, 30\\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 4$ MPI processes.
Figure 13: 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 $\alpha \in \{0.1, 1\}$, $r \in \{1\% , 5\% , 10\% \}$ and $o = 20\%$. Bottom part: we used $\alpha = 1$, $r = 1\%$ and $o \in \{5\% , 10\% , 20\% , 30\% \}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.
Figure 14: *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 $o \in \{5\%, 10\%, 20\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.
Figure 15: **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 $o \in \{5\%, 10\% , 20\% , 30\% \}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.
Figure 16: 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 $o \in \{5\%, 10\%, 20\%, 30\%\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.
Figure 17: **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 $o \in \{5\%, 10\%, 20\%, 30\\}$. Solid lines show average performance, and dashed lines show worst and best performance, over 10 runs (per algorithm). $K = 16$ MPI processes.
Figure 18: 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 $o \in \{5\%, 10\%, 20\%, 30\\%\}$. 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 $A, B \in \mathbb{R}^{n \times n}$, 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.

![Diagram](image)

Figure 19: 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 19. 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 20-24 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 20: 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 21: **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 22: **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 23: **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 24: **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 two Neural Network training tasks on the MNIST\(^4\) data set. More specifically, we show the training and testing accuracy of the methods for all batch sizes (|\(S\)| ∈ \{32, 64, ..., 4096\}). The two tasks are summarized in Table 3, and the details are explained below.

<table>
<thead>
<tr>
<th>Network</th>
<th>Type</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>Convolutional</td>
<td>Conv(32 features,5x5 patch) - Max pool (2x2) - Conv(64 features,5x5 patch) - Max pool (2x2) - Fully (1024) Softmax cross entropy loss function</td>
</tr>
<tr>
<td>Regression</td>
<td>Feed-forward</td>
<td>NN with no hidden layers (\ell_2) loss function (convex problem)</td>
</tr>
</tbody>
</table>

Table 3: Structure of Neural Networks

- **DNN:** This network consists of two convolutional layers and one fully connected layer. The first convolutional layer has 32 features for each 5 × 5 patch, and the second convolutional layer has 64 features for each 5 × 5 patch. Every convolutional layer is followed by a 2 × 2 max pool.

- **Regression:** This problem was a convex problem. It is a simple Neural Network with no hidden layers and \(\ell_2\) loss function.

For the experiments in this section (Figures 25 and 26), we ran the following four methods:

- (LFBGS) multi-batch L-BFGS: Algorithm 1 with standard scaling (\(\gamma_k I\), where \(\gamma_k = \frac{s_k^T y_k - s_k^T y_{k-1}}{y_k^T y_{k-1}}\)) of the initial Hessian approximation Nocedal and Wright (1999)

- (LBFGS2) multi-batch L-BFGS: Algorithm 1 with \(\alpha I\) initial scaling

- (Adam) Kingma and Ba (2014)

- (Adagrad) Duchi et al. (2011)

- (Adadelta) Zeiler (2012)

- (SGD)

In Figures 25 and 26 we show the evolution of training and testing accuracy for different batch sizes for the two different Neural Network training tasks. The second and fourth rows of the figures are zoomed in versions of the first and third rows, respectively, at the high accuracy level.

Figure 25: DNN - MNIST. Comparison of the multi-batch L-BFGS variants (LBFGS, LBFGS2), Adam, Adagrad, Adadelta and SGD. First and third rows: Evolution of the training and testing accuracy for different batch sizes. Second and fourth rows: Zoomed in versions of the first and third rows, respectively, for high accuracy.
Figure 26: Regression - MNIST. Comparison of the multi-batch L-BFGS variants (LBFGS, LBFGS2), Adam, Adagrad, Adadelta and SGD. First and third rows: Evolution of the training and testing accuracy for different batch sizes. Second and fourth rows: Zoomed in versions of the first and third rows, respectively, for high accuracy.
References


