A Progressive Batching L-BFGS Method for Machine Learning

Raghu Bollapragada 1  Dheevatsa Mudigere 2  Jorge Nocedal 1  Hao-Jun Michael Shi 1  Ping Tak Peter Tang 3

Abstract

The standard L-BFGS method relies on gradient approximations that are not dominated by noise, so that search directions are descent directions, the line search is reliable, and quasi-Newton updating yields useful quadratic models of the objective function. All of this appears to call for a full batch approach, but since small batch sizes give rise to faster algorithms with better generalization properties, L-BFGS is currently not considered an algorithm of choice for large-scale machine learning applications. One need not, however, choose between the two extremes represented by the full batch or highly stochastic regimes, and may instead follow a progressive batching approach in which the sample size increases during the course of the optimization. In this paper, we present a new version of the L-BFGS algorithm that combines three basic components — progressive batching, a stochastic line search, and stable quasi-Newton updating — and that performs well on training logistic regression and deep neural networks. We provide supporting convergence theory for the method.

1. Introduction

The L-BFGS method (Liu & Nocedal, 1989) has traditionally been regarded as a batch method in the machine learning community. This is because quasi-Newton algorithms need gradients of high quality in order to construct useful quadratic models and perform reliable line searches. These algorithmic ingredients can be implemented, it seems, only by using very large batch sizes, resulting in a costly iteration that makes the overall algorithm slow compared with stochastic gradient methods (Robbins & Monro, 1951).

Even before the resurgence of neural networks, many researchers observed that a well-tuned implementation of the stochastic gradient (SG) method was far more effective on large-scale logistic regression applications than the batch L-BFGS method, even when taking into account the advantages of parallelism offered by the use of large batches. The preeminence of the SG method (and its variants) became more pronounced with the advent of deep neural networks, and some researchers have speculated that SG is endowed with certain regularization properties that are essential in the minimization of such complex nonconvex functions (Hardt et al., 2015; Keskar et al., 2016).

In this paper, we postulate that the most efficient algorithms for machine learning may not reside entirely in the highly stochastic or full batch regimes, but should employ a progressive batching approach in which the sample size is initially small, and is increased as the iteration progresses. This view is consistent with recent numerical experiments on training various deep neural networks (Smith et al., 2017; Goyal et al., 2017), where the SG method, with increasing sample sizes, yields similar test loss and accuracy as the standard (fixed mini-batch) SG method, while offering significantly greater opportunities for parallelism.

Progressive batching algorithms have received much attention recently from a theoretical perspective. It has been shown that they enjoy complexity bounds that rival those of the SG method (Byrd et al., 2012), and that they can achieve a fast rate of convergence (Friedlander & Schmidt, 2012). The main appeal of these methods is that they inherit the efficient initial behavior of the SG method, offer greater opportunities to exploit parallelism, and allow for the incorporation of second-order information. The latter can be done efficiently via quasi-Newton updating.

An integral part of quasi-Newton methods is the line search, which ensures that a convex quadratic model can be constructed at every iteration. One challenge that immediately arises is how to perform this line search when the objective function is stochastic. This is an issue that has not received sufficient attention in the literature, where stochastic line searches have been largely dismissed as inappropriate. In this paper, we take a step towards the development of stochastic line searches for machine learning by study-
ing a key component, namely the initial estimate in the one-dimensional search. Our approach, which is based on statistical considerations, is designed for an Armijo-style backtracking line search.

1.1. Literature Review

Progressive batching (sometimes referred to as dynamic sampling) has been well studied in the optimization literature, both for stochastic gradient and subsampled Newton-type methods (Byrd et al., 2012; Friedlander & Schmidt, 2012; Cartis & Scheinberg, 2015; Pasupathy et al., 2015; Roosta-Khorasani & Mahoney, 2016a;b; Bollapragada et al., 2016; 2017; De et al., 2017). Friedlander and Schmidt (2012) introduced theoretical conditions under which a progressive batching SG method converges linearly for finite sum problems, and experimented with a quasi-Newton adaptation of their algorithm. Byrd et al. (2012) proposed a progressive batching strategy, based on a norm test, that determines when to increase the sample size; they established linear convergence and computational complexity bounds in the case when the batch size grows geometrically. More recently, Bollapragada et al. (2017) introduced a batch control mechanism based on an inner product test that improves upon the norm test mentioned above.

There has been a renewed interest in understanding the generalization properties of small-batch and large-batch methods for training neural networks; see (Keskar et al., 2016; Dinh et al., 2017; Goyal et al., 2017; Hoffer et al., 2017). Keskar et al. (2016) empirically observed that large-batch methods converge to solutions with inferior generalization properties; however, Goyal et al. (2017) showed that large-batch methods can match the performance of small-batch methods when a warm-up strategy is used in conjunction with scaling the step length by the same factor as the batch size. Hoffer et al. (2017) and You et al. (2017) also explored larger batch sizes and steplengths to reduce the number of updates necessary to train the network. All of these studies naturally led to an interest in progressive batching techniques. Smith et al. (2017) showed empirically that increasing the sample size and decaying the steplength are quantitatively equivalent for the SG method; hence, steplength schedules could be directly converted to batch size schedules. This approach was parallelized by Devarakonda et al. (2017). De et al. (2017) presented numerical results with a progressive batching method that employs the norm test. Balles et al. (2016) proposed an adaptive dynamic sample size scheme and couples the sample size with the steplength.

Stochastic second-order methods have been explored within the context of convex and non-convex optimization; see (Schraudolph et al., 2007; Sohl-Dickstein et al., 2014; Mokhtari & Ribeiro, 2015; Berahas et al., 2016; Byrd et al., 2016; Keskar & Berahas, 2016; Curtis, 2016; Berahas & Takáč, 2017; Zhou et al., 2017). Schraudolph et al. (2007) ensured stability of quasi-Newton updating by computing gradients using the same batch at the beginning and end of the iteration. Since this can potentially double the cost of the iteration, Berahas et al. (2016) proposed to achieve gradient consistency by computing gradients based on the overlap between consecutive batches; this approach was further tested by Berahas and Takac (2017). An interesting approach introduced by Martens and Grosse (2015; 2016) approximates the Fisher information matrix to scale the gradient; a distributed implementation of their K-FAC approach is described in (Ba et al., 2016). Another approach approximately computes the inverse Hessian by using the Neumann power series representation of matrices (Krishnan et al., 2017).

1.2. Contributions

This paper builds upon three algorithmic components that have recently received attention in the literature — progressive batching, stable quasi-Newton updating, and adaptive steplength selection. It advances their design and puts them together in a novel algorithm with attractive theoretical and computational properties.

The cornerstone of our progressive batching strategy is the mechanism proposed by Bollapragada et al. (2017) in the context of first-order methods. We extend their inner product control test to second-order algorithms, something that is delicate and leads to a significant modification of the original procedure. Another main contribution of the paper is the design of an Armijo-style backtracking line search where the initial steplength is chosen based on statistical information gathered during the course of the iteration. We show that this steplength procedure is effective on a wide range of applications, as it leads to well scaled steps and allows for the BFGS update to be performed most of the time, even for nonconvex problems. We also test two techniques for ensuring the stability of quasi-Newton updating, and observe that the overlapping procedure described by Berahas et al. (2016) is more efficient than a straightforward adaptation of classical quasi-Newton methods (Schraudolph et al., 2007).

We report numerical tests on large-scale logistic regression and deep neural network training tasks that indicate that our method is robust and efficient, and has good generalization properties. An additional advantage is that the method requires almost no parameter tuning, which is possible due to the incorporation of second-order information. All of this suggests that our approach has the potential to become one of the leading optimization methods for training deep neural networks. In order to achieve this, the algorithm must be optimized for parallel execution, something that was only briefly explored in this study.
2. A Progressive Batching Quasi-Newton Method

The problem of interest is
\[
\min_{x \in \mathbb{R}^d} F(x) = \int f(x; z, y) dP(z, y),
\]
(1)
where \( f \) is the composition of a prediction function (parametrized by \( x \)) and a loss function, and \((z, y)\) are random input-output pairs with probability distribution \( P(z, y) \). The associated empirical risk problem consists of minimizing
\[
R(x) = \frac{1}{N} \sum_{i=1}^{N} f(x; z^i, y^i) \triangleq \frac{1}{N} \sum_{i=1}^{N} F_i(x),
\]
where we define \( F_i(x) = f(x; z^i, y^i) \). A stochastic quasi-Newton method is given by
\[
x_{k+1} = x_k - \alpha_k H_k g_k^{S_k},
\]
(2)
where the batch (or subsampled) gradient is given by
\[
g_k^{S_k} = \nabla F_{S_k}(x_k) \triangleq \frac{1}{|S_k|} \sum_{i \in S_k} \nabla F_i(x_k),
\]
(3)
the set \( S_k \subset \{1, 2, \cdots \} \) indexes data points \((y^i, z^i)\) sampled from the distribution \( P(z, y) \), and \( H_k \) is a positive definite quasi-Newton matrix. We now discuss each of the components of the new method.

2.1. Sample Size Selection

The proposed algorithm has the form (2)-(3). Initially, it utilizes a small batch size \(|S_k|\), and increases it gradually in order to attain a fast local rate of convergence and permit the use of second-order information. A challenging question is to determine when, and by how much, to increase the batch size \(|S_k|\) over the course of the optimization procedure based on observed gradients — as opposed to using prescribed rules that depend on the iteration number \( k \).

We propose to build upon the strategy introduced by Bollapragada et al. (2017) in the context of first-order methods. Their inner product test determines a sample size such that the search direction is a descent direction with high probability. A straightforward extension of this strategy to the quasi-Newton setting is not appropriate since requiring only that a stochastic quasi-Newton search direction be a descent direction with high probability would underutilize the curvature information contained in the search direction.

We would like, instead, for the search direction \( d_k = -H_k g_k^{S_k} \) to make an acute angle with the true quasi-Newton search direction \(-H_k \nabla F(x_k)\), with high probability. Although this does not imply that \( d_k \) is a descent direction for \( F \), this will normally be the case for any reasonable quasi-Newton matrix.

To derive the new inner product quasi-Newton (IPQN) test, we first observe that the stochastic quasi-Newton search direction makes an acute angle with the true quasi-Newton direction in expectation, i.e.,
\[
\mathbb{E}_k \left[ (H_k \nabla F(x_k))^T (H_k g_k^{S_k}) \right] = \|H_k \nabla F(x_k)\|^2,
\]
(4)
where \( \mathbb{E}_k \) denotes the conditional expectation at \( x_k \). We must, however, control the variance of this quantity to achieve our stated objective. Specifically, we select the sample size \(|S_k|\) such that the following condition is satisfied:
\[
\mathbb{E}_k \left[ \left( (H_k \nabla F(x_k))^T (H_k g_k^{S_k}) - \|H_k \nabla F(x_k)\|^2 \right)^2 \right] \leq \theta^2 \|H_k \nabla F(x_k)\|^4,
\]
(5)
for some \( \theta > 0 \). The left hand side of (5) is difficult to compute but can be bounded by the true variance of individual search directions, i.e.,
\[
\mathbb{E}_k \left[ \left( (H_k \nabla F(x_k))^T (H_k g_k^{S_k}) - \|H_k \nabla F(x_k)\|^2 \right)^2 \right] \leq \theta^2 \|H_k \nabla F(x_k)\|^4,
\]
(6)
where \( g_k^i = \nabla F_i(x_k) \). This test involves the true expected gradient and variance, but we can approximate these quantities with sample gradient and variance estimates, respectively, yielding the practical inner product quasi-Newton test:
\[
\text{Var}_{i \in S_k^c} \left( (g_k^i)^T H_k^2 g_k^{S_k} \right) \leq \theta^2 \left( \|H_k g_k^{S_k}\|^4 \right),
\]
(7)
where \( S_k^c \subset S_k \) is a subset of the current sample (batch), and the variance term is defined as
\[
\sum_{i \in S_k^c} \left( (g_k^i)^T H_k^2 g_k^{S_k} - \left( \|H_k g_k^{S_k}\|^2 \right) \right)^2 \leq \theta^2 \left( \|H_k g_k^{S_k}\|^4 \right),
\]
(8)
The variance (8) may be computed using just one additional Hessian vector product of \( H_k \) with \( H_k g_k^{S_k} \). Whenever condition (7) is not satisfied, we increase the sample size \(|S_k|\).

In order to estimate the increase that would lead to a satisfaction of (7), we reason as follows. If we assume that new sample \(|S_k|\) is such that
\[
\|H_k g_k^{S_k}\| \approx \|H_k g_k^{S_k}\|,
\]
and similarly for the variance estimate, then a simple computation shows that a lower bound on the new sample size
is
\[ |\tilde{S}_k| \geq \frac{\text{Var}_{i \in S_k} \left( (g_k)_i^T H_k^2 g_k^{S_k} \right)}{\theta_1^2 \|H_k g_k^{S_k}\|^4} \triangleq b_k. \quad (9) \]

In our implementation of the algorithm, we set the new sample size as \(|S_{k+1}| = \lfloor b_k \rfloor\). When the sample approximation of \(F(x_k)\) is not accurate, which can occur when \(|S_k|\) is small, the progressive batching mechanism just described may not be reliable. In this case we employ the moving window technique described in Section 4.2 of Bollapragada et al. (2017), to produce a sample estimate of \(\nabla F(x_k)\).

2.2. The Line Search

In deterministic optimization, line searches are employed to ensure that the step is not too short and to guarantee sufficient decrease in the objective function. Line searches are particularly important in quasi-Newton methods since they ensure robustness and efficiency of the iteration with little additional cost.

In contrast, stochastic line searches are poorly understood and rarely employed in practice because they must make decisions based on sample function values
\[ F_{S_k}(x) = \frac{1}{|S_k|} \sum_{i \in S_k} F_i(x), \quad (10) \]
which are noisy approximations to the true objective \(F\). One of the key questions in the design of a stochastic line search is how to ensure, with high probability, that there is a decrease in the true function when one can only observe stochastic approximations \(F_{S_k}(x)\). We address this question by proposing a formula for the step size \(\alpha_k\) that controls possible increases in the true function. Specifically, the first trial steplength in the stochastic backtracking line search is computed so that the predicted decrease in the expected function value is sufficiently large, as we now explain.

Using Lipschitz continuity of \(\nabla F(x)\) and taking conditional expectation, we can show the following inequality
\[ \mathbb{E}_k [F_{k+1}] \leq F_k - \alpha_k \nabla F(x_k)^T H_k^{1/2} W_k H_k^{1/2} \nabla F(x_k) \quad (11) \]
where
\[ W_k = \left( I - \frac{L \alpha_k}{2} \right) \left( 1 + \frac{\text{Var}_k \left( H_k g_k^1 \right)}{|S_k| \|H_k \nabla F(x_k)\|^2} \right) H_k, \]
\[ \text{Var}_k \left( H_k g_k^1 \right) = \mathbb{E}_k \left[ \|H_k g_k^1 - H_k \nabla F(x_k)\|^2 \right], \]
and \(L\) is the Lipschitz constant. The proof of (11) is given in the supplement.

The only difference in (11) between the deterministic and stochastic quasi-Newton methods is the additional variance term in the matrix \(W_k\). To obtain decrease in the function value in the deterministic case, the matrix \(I - \frac{L \alpha_k}{2} H_k\) must be positive definite, whereas in the stochastic case the matrix \(W_k\) must be positive definite to yield a decrease in \(F\) in expectation. In the deterministic case, for a reasonably good quasi-Newton matrix \(H_k\), one expects that \(\alpha_k = 1\) will result in a decrease in the function, and therefore the initial trial steplength parameter should be chosen to be 1. In the stochastic case, the initial trial value
\[ \hat{\alpha}_k = \left( 1 + \frac{\text{Var}_k \left( H_k g_k^1 \right)}{|S_k| \|H_k \nabla F(x_k)\|^2} \right)^{-1} \quad (12) \]
will result in decrease in the expected function value. However, since formula (12) involves the expensive computation of the individual matrix-vector products \(H_k g_k^i\), we approximate the variance-bias ratio as follows:
\[ \tilde{\alpha}_k = \left( 1 + \frac{\text{Var}_k \left( g_k^1 \right)}{|S_k| \|H_k \nabla F(x_k)\|^2} \right)^{-1}, \quad (13) \]
where \(\text{Var}_k \left( g_k^1 \right) = \mathbb{E}_k \left[ \|g_k^1 - \nabla F(x_k)\|^2 \right]\). In our practical implementation, we estimate the population variance and gradient with the sample variance and gradient, respectively, yielding the initial steplength
\[ \alpha_k = \left( 1 + \frac{\text{Var}_{i \in S_k} \left( g_k^i \right)}{|S_k| \left\| g_k^{S_k} \right\|^T} \right)^{-1}, \quad (14) \]
where
\[ \text{Var}_{i \in S_k} \left( g_k^i \right) = \frac{1}{|S_k| - 1} \sum_{i \in S_k} \left\| g_k^i - g_k^{S_k} \right\|^2 \quad (15) \]
and \(S_k^c \subseteq S_k\). With this initial value of \(\alpha_k\) in hand, our algorithm performs a backtracking line search that aims to satisfy the Armijo condition
\[ F_{S_k}(x_k - \alpha_k H_k g_k^{S_k}) \leq F_{S_k}(x_k) - c_1 \alpha_k (g_k^{S_k})^T H_k g_k^{S_k}, \quad (16) \]
where \(c_1 > 0\).

2.3. Stable Quasi-Newton Updates

In the BFGS and L-BFGS methods, the inverse Hessian approximation is updated using the formula
\[ H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \]
\[ \rho_k = (g_k^T s_k)^{-1} \]
\[ V_k = I - \rho_k y_k s_k^T \quad (17) \]
where \(s_k = x_{k+1} - x_k\) and \(y_k\) is the difference in the gradients at \(x_{k+1}\) and \(x_k\). When the batch changes from one iteration to the next \((S_k+1 \neq S_k)\), it is not obvious how
$y_k$ should be defined. It has been observed that when $y_k$ is computed using different samples, the updating process may be unstable, and hence it seems natural to use the same sample at the beginning and at the end of the iteration (Schraudolph et al., 2007), and define
\[
y_k = g^T_k S_k - g^T_k S_{k+1}.
\] (18)
However, this requires that the gradient be evaluated twice for every batch $S_k$ at $x_k$ and $x_{k+1}$. To avoid this additional cost, Berahas et al. (2016) propose to use the overlap between consecutive samples in the gradient differencing. If we denote this overlap as $O_k = S_k \cap S_{k+1}$, then one defines
\[
y_k = g^T_k O_k - g^T_k O_{k+1}.
\] (19)
This requires no extra computation since the two gradients in this expression are subsets of the gradients corresponding to the samples $S_k$ and $S_{k+1}$. The overlap should not be too small to avoid differencing noise, but this is easily achieved in practice. We test both formulas for $y_k$ in our implementation of the method; see Section 4.

2.4. The Complete Algorithm

The pseudocode of the progressive batching L-BFGS method is given in Algorithm 1. Observe that the limited memory Hessian approximation $H_k$ in Line 8 is independent of the choice of the sample $S_k$. Specifically, $H_k$ is defined by a collection of curvature pairs $\{(s_j, y_j)\}$, where the most recent pair is based on the sample $S_{k-1}$; see Line 14. For the batch size control test (7), we choose $\theta = 0.9$ in the logistic regression experiments, and $\theta$ is a tunable parameter chosen in the interval $[0.9, 3]$ in the neural network experiments. The constant $c_1$ in (16) is set to $c_1 = 10^{-4}$. For L-BFGS, we set the memory as $m = 10$. We skip the quasi-Newton update if the following curvature condition is not satisfied:
\[
y^T_k s_k > \varepsilon \|s_k\|^2, \quad \text{with} \quad \varepsilon = 10^{-2}.
\] (20)
The initial Hessian matrix $H_0$ in the L-BFGS recursion at each iteration is chosen as $\gamma_k I$ where $\gamma_k = y^T_k s_k / y^T_k y_k$.

3. Convergence Analysis

We now present convergence results for the proposed algorithm, both for strongly convex and nonconvex objective functions. Our emphasis is in analyzing the effect of progressive sampling, and therefore, we follow common practice and assume that the steplength in the algorithm is fixed ($\alpha_k = \alpha$), and that the inverse L-BFGS matrix $H_k$ has bounded eigenvalues, i.e.,
\[
\Lambda_1 I \preceq H_k \preceq \Lambda_2 I.
\] (21)
This assumption can be justified both in the convex and nonconvex cases under certain conditions; see (Berahas et al., 2016). We assume that the sample size is controlled by the exact inner product quasi-Newton test (6). This test is designed for efficiency, and in rare situations could allow for the generation of arbitrarily long search directions. To prevent this from happening, we introduce an additional control on the sample size $|S_k|$, by extending (to the quasi-Newton setting) the orthogonality test introduced in (Bollapragada et al., 2017). This additional requirement states that the current sample size $|S_k|$ is acceptable only if
\[
\mathbb{E}_k \left[ \frac{\|H_k y^T_k \|}{\|H_k y^T_k (F(x_k))\|} \|H_k \nabla F(x_k)\|^2 \right] \leq \nu^2 \|H_k \nabla F(x_k)\|^2,
\] (22)
for some given $\nu > 0$.

We now establish linear convergence when the objective is strongly convex.

Theorem 3.1. Suppose that $F$ is twice continuously differentiable and that there exist constants $0 < \mu \leq L$ such that
\[
\mu I \leq \nabla^2 F(x) \leq LI, \quad \forall x \in \mathbb{R}^d.
\] (23)
Let \( \{x_k\} \) be generated by iteration (2), for any \( x_0 \), where 
\[ |S_k| \] is chosen by the (exact variance) inner product quasi-
Newton test (6). Suppose that the orthogonality condition 
(22) holds at every iteration, and that the matrices \( H_k \) sat-
ify (21). Then, if 
\[
\alpha_k = \alpha \leq \frac{1}{(1 + \theta^2 + \nu^2)L_2},
\]
we have that 
\[
\mathbb{E}[F(x_k) - F(x^*)] \leq \rho^k(F(x_0) - F(x^*)),
\]
where \( x^* \) denotes the minimizer of \( F \), \( \rho = 1 - \mu L_2 \alpha \), and 
\( \mathbb{E} \) denotes the total expectation.

The proof of this result is given in the supplement. We now
consider the case when \( F \) is nonconvex and bounded below.

**Theorem 3.2.** Suppose that \( F \) is twice continuously differ-
entiable and bounded below, and that there exists a constant 
\( L > 0 \) such that 
\[
\nabla^2 F(x) \preceq LI, \quad \forall x \in \mathbb{R}^d.
\]

Let \( \{x_k\} \) be generated by iteration (2), for any \( x_0 \), where 
\[ |S_k| \] is chosen so that (6) and (22) are satisfied, and suppose 
that (21) holds. Then, if \( \alpha_k \) satisfies (24), we have 
\[
\lim_{k \to \infty} \mathbb{E}[\|\nabla F(x_k)\|^2] \to 0.
\]
Moreover, for any positive integer \( T \) we have that 
\[
\min_{0 \leq k \leq T-1} \mathbb{E}[\|\nabla F(x_k)\|^2] \leq \frac{2}{\alpha TA_1}(F(x_0) - F_{\min}),
\]
where \( F_{\min} \) is a lower bound on \( F \) in \( \mathbb{R}^d \).

The proof is given in the supplement. This result shows that 
the sequence of gradients \( \{\|\nabla F(x_k)\|\} \) converges to zero 
in expectation, and establishes a global sublinear rate of 
convergence of the smallest gradients generated after every 
\( T \) steps.

### 4. Numerical Results

In this section, we present numerical results for the proposed 
algorithm, which we refer to as PBQN for the **Progressive 
Batching Quasi-Newton** algorithm.

#### 4.1. Experiments on Logistic Regression Problems

We first test our algorithm on binary classification problems 
where the objective function is given by the logistic loss 
with \( \ell_2 \) regularization:
\[
R(x) = \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-z_i^T x^T y^i)) + \frac{\lambda}{2} \|x\|^2,
\]
with \( \lambda = 1/N \). We consider the 8 datasets listed in the 
supplement. An approximation \( R^* \) of the optimal function 
value is computed for each problem by running the full batch 
L-BFGS method until \( \|\nabla R(x_k)\|_\infty \leq 10^{-8} \). Training error 
is defined as \( R(x_k) - R^* \), where \( R(x_k) \) is evaluated over the 
training set; test loss is evaluated over the test set without 
the \( \ell_2 \) regularization term.

We tested two options for computing the curvature vector 
\( y_k \) in the PBQN method: the multi-batch (MB) approach 
(19) with 25% sample overlap, and the full overlap (FO) 
approach (18). We set \( \theta = 0.9 \) in (7), chose \( |S_0| = 512 \), and 
set all other parameters to the default values given in 
Section 2. Thus, none of the parameters in our PBQN method 
were tuned for each individual dataset. We compared our 
algorithm against two other methods: (i) Stochastic gradient 
(SG) with a batch size of 1; (ii) SVRG (Johnson & Zhang, 
2013) with the inner loop length set to \( N \). The steplength 
for SG and SVRG is constant and tuned for each problem 
(\( \alpha_k \equiv \alpha = 2^j \), for \( j \in \{-10, -9, ..., 9, 10\} \)) so as to give 
best performance.

In Figures 1 and 2 we present results for two datasets, spam 
and covtype; the rest of the results are given in the 
supplement. The horizontal axis measures the number of 
full gradient evaluations, or equivalently, the number of 
times that \( N \) component gradients \( \nabla F \) were evaluated. The 
left-most figure reports the long term trend over 100 gradient 
evaluations, while the rest of the figures zoom into the first 
10 gradient evaluations to show the initial behavior of the 
methods. The vertical axis measures training error, test loss, 
and test accuracy, respectively, from left to right.

The proposed algorithm competes well for these two datasets in terms of training error, test loss and test accuracy, 
and decreases these measures more evenly than the SG and 
SVRG. Our numerical experience indicates that formula 
(14) is quite effective at estimating the steplength parameter, 
as it is accepted by the backtracking line search for most 
iterations. As a result, the line search computes very few 
additional function values.

It is interesting to note that SVRG is not as efficient in the 
initial epochs compared to PBQN or SG, when measured 
either in terms of test loss and test accuracy. The training error 
for SVRG decreases rapidly in later epochs but this rapid 
 improvement is not observed in the test loss and accuracy.
Neither the PBQN nor SVRG significantly outperforms the 
other across all datasets tested in terms of training error, as 
observed in the supplement.

Our results indicate that defining the curvature vector using 
the MB approach is preferable to using the FB approach. 
The number of iterations required by the PBQN method is 
significantly smaller compared to the SG method, suggesting 
the potential efficiency gains of a parallel implementa-
Figure 1. spam dataset: Performance of the progressive batching L-BFGS method (PBQN), with multi-batch (25% overlap) and full-overlap approaches, and the SG and SVRG methods.

Figure 2. covertype dataset: Performance of the progressive batching L-BFGS methods, with multi-batch (25% overlap) and full-overlap approaches, and the SG and SVRG methods.

4.2. Results on Neural Networks

We have performed a preliminary investigation into the performance of the PBQN algorithm for training neural networks. As is well-known, the resulting optimization problems are quite difficult due to the existence of local minimizers, some of which generalize poorly. Thus our first requirement when applying the PBQN method was to obtain as good generalization as SG, something we have achieved.

Our investigation into how to obtain fast performance is, however, still underway for reasons discussed below. Nevertheless, our results are worth reporting because they show that our line search procedure is performing as expected, and that the overall number of iterations required by the PBQN method is small enough so that a parallel implementation could yield state-of-the-art results, based on the theoretical performance model detailed in the supplement.

We compared our algorithm, as described in Section 2, against SG and Adam (Kingma & Ba, 2014). It has taken many years to design regularization techniques and heuristics that greatly improve the performance of the SG method for deep learning (Srivastava et al., 2014; Ioffe & Szegedy, 2015). These include batch normalization and dropout, which (in their current form) are not conducive to the PBQN approach due to the need for gradient consistency when evaluating the curvature pairs in L-BFGS. Therefore, we do not implement batch normalization and dropout in any of the methods tested, and leave the study of their extension to the PBQN setting for future work.

We consider three network architectures: (i) a small convolutional neural network on CIFAR-10 (C) (Krizhevsky, 2009), (ii) an AlexNet-like convolutional network on MNIST and CIFAR-10 (A1, A2, respectively) (LeCun et al., 1998; Krizhevsky et al., 2012), and (iii) a residual network (ResNet18) on CIFAR-10 (R) (He et al., 2016). The network architecture details and additional plots are given in the supplement. All of these networks were implemented in PyTorch (Paszke et al., 2017). The results for the CIFAR-10 AlexNet and CIFAR-10 ResNet18 are given in Figures 3 and 4, respectively. We report results both against the total number of iterations and the total number of gradient evaluations. Table 1 shows the best test accuracies attained by each of the four methods over the various networks.

In all our experiments, we initialize the batch size as \( |S_0| = 512 \) in the PBQN method, and fix the batch size to \( |S_k| = 128 \) for SG and Adam. The parameter \( \Theta \) given in (7), which controls the batch size increase in the PBQN method, was tuned lightly by choosing among the 3 values: 0.9, 2, 3. SG and Adam are tuned using a development-based decay (dev-decay) scheme, which track the best validation loss at each epoch and reduces the steplength by a constant factor \( \delta \) if the validation loss does not improve after \( e \) epochs.

We observe from our results that the PBQN method achieves a similar test accuracy as SG and Adam, but requires more gradient evaluations. Improvements in performance can be obtained by ensuring that the PBQN method exerts a finer control on the sample size in the small batch regime — something that requires further investigation. Nevertheless, the small number of iterations required by the PBQN method, together with the fact that it employs larger batch sizes than...
SG during much of the run, suggests that a distributed version similar to a data-parallel distributed implementation of the SG method (Chen et al., 2016; Das et al., 2016) would lead to a highly competitive method.

Similar to the logistic regression case, we observe that the steplength computed via (14) is almost always accepted by the Armijo condition, and typically lies within (0, 1, 1). Once the algorithm has trained for a significant number of iterations using full-batch, the algorithm begins to overfit on the training set, resulting in worsened test loss and accuracy, as observed in the graphs.

5. Final Remarks

Several types of quasi-Newton methods have been proposed in the literature to address the challenges arising in machine learning. Some of these methods operate in the purely stochastic setting (which makes quasi-Newton updating difficult) or in the purely batch regime (which leads to generalization problems). We believe that progressive batching is the right context for designing an L-BFGS method that has good generalization properties, does not expose any free parameters, and has fast convergence. The advantages of our approach are clearly seen in logistic regression experiments. To make the new method competitive with SG and Adam for deep learning, we need to improve several of its components. This includes the design of a more robust progressive batching mechanism, the redesign of batch normalization and dropout heuristics to improve the generalization performance of our method for training larger networks, and most importantly, the design of a parallelized implementation that takes advantage of the higher granularity of each iteration. We believe that the potential of the proposed approach as an alternative to SG for deep learning is worthy of further investigation.

Acknowledgements

We thank Albert Berahas for his insightful comments regarding multi-batch L-BFGS and probabilistic line searches, as well as for his useful feedback on earlier versions of the manuscript. We also thank the anonymous reviewers for their useful feedback. Bollapragada is supported by DOE award DE-FG02-87ER25047. Nocedal is supported by NSF award DMS-1620070. Shi is supported by Intel grant SP0036122.

References

Ba, J., Grosse, R., and Martens, J. Distributed second-order optimization using Kronecker-factored approximations.
A Progressive Batching L-BFGS Method for Machine Learning

2016.


A Progressive Batching L-BFGS Method for Machine Learning


