A Distributed Quasi-Newton Algorithm for Empirical Risk Minimization with Nonsmooth Regularization

Ching-pei Lee
Department of Computer Sciences
University of Wisconsin-Madison
Madison, Wisconsin
ching-pei@cs.wisc.edu

Cong Han Lim
Wisconsin Institute for Discovery
University of Wisconsin-Madison
Madison, Wisconsin
clim9@wisc.edu

Stephen J. Wright
Department of Computer Sciences
University of Wisconsin-Madison
Madison, Wisconsin
swright@cs.wisc.edu

ABSTRACT

We propose a communication- and computation-efficient distributed optimization algorithm using second-order information for solving ERM problems with a nonsmooth regularization term. Current second-order and quasi-Newton methods for this problem either do not work well in the distributed setting or work only for specific regularizers. Our algorithm uses successive quadratic approximations, and we describe how to maintain an approximation of the Hessian and solve subproblems efficiently in a distributed manner. The proposed method enjoys global linear convergence for a broad range of non-strongly convex problems that includes the most commonly used ERMs, thus requiring lower communication complexity. It also converges on non-convex problems, so has the potential to be used on applications such as deep learning. Initial computational results on convex problems demonstrate that our method significantly improves on communication cost and running time over the current state-of-the-art methods.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; **Distributed algorithms**;

KEYWORDS

Distributed optimization, Empirical risk minimization, Nonsmooth optimization, Regularized optimization, Variable metrics, Quasi-Newton, Proximal method, Inexact method

ACM Reference Format:
Ching-pei Lee, Cong Han Lim, and Stephen J. Wright. 2018. A Distributed Quasi-Newton Algorithm for Empirical Risk Minimization with Nonsmooth Regularization. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220075

KDD '18, August 19–23,2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220075

1 INTRODUCTION

We consider solving the following regularized problem in a distributed manner:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \quad F(\boldsymbol{w}) \coloneqq f(X^T\boldsymbol{w}) + g(\boldsymbol{w}), \tag{1}$$

where X is a d by n real-valued matrix, g is a convex, closed, and extended-valued proper function that can be nondifferentiable, and f is a differentiable function whose gradient is Lipschitz continuous with parameter $L > 0$. Each column of X represents a single data point or instance, and we assume that the set of data points is partitioned and spread across K machines (i.e. distributed *instance-wise*). We can write X as

$$X \coloneqq [X_1, X_2, \ldots, X_K],$$

where X_k is stored exclusively on the kth machine. We further assume that f shares the same block-separable structure and can be written as follows:

$$f\left(X^T\boldsymbol{w}\right) = \sum_{k=1}^{K} f_k\left(X_k^T\boldsymbol{w}\right).$$

Unlike our instance-wise setting, some existing works consider the *feature-wise* partition setting, under which X is partitioned by rows rather than columns. Although the feature-wise setting is a simpler one for algorithm design when g is separable, storage of different features on different machines is often impractical.

The bottleneck in performing distributed optimization is often the high cost of communication between machines. For (1), the time required to retrieve X_k over a network can greatly exceed the time needed to compute f_k or its gradient with locally stored X_k. Moreover, we incur a delay at the beginning of each round of communication due to the overhead of establishing connections between machines. This latency prevents many efficient single-core algorithms such as coordinate descent (CD) and stochastic gradient and their asynchronous parallel variants from being employed in large-scale distributed computing setups. Thus, a key aim of algorithm design for distributed optimization is to improve the communication efficiency while keeping the computational cost affordable. Batch methods are preferred in this context, because fewer rounds of communication occur in distributed batch methods.

When F is differentiable, many efficient batch methods can be used directly in distributed environments to solve (1). For example, Nesterov's accelerated gradient (AG) [16] enjoys low iteration complexity, and since each iteration of AG only requires one round of communication to compute the new gradient, it also has good communication complexity. Although its supporting theory is not particularly strong, the limited-memory BFGS (LBFGS) method [13] is popular among practitioners of distributed optimization. It is the default algorithm for solving ℓ_2-regularized smooth ERM problems in Apache Spark's distributed machine learning library [14], as it is empirically much faster than AG (see, for example, the experiments in Wang et al. [22]). Other modified batch methods that utilize the Hessian of the objective in various ways are also communication-efficient under their own additional assumptions [7, 12, 20, 26, 28].

When g is nondifferentiable, neither LBFGS nor Newton's method can be applied directly. Leveraging curvature information from f can still be beneficial in this setting. For example, the orthant-wise quasi-Newton method OWLQN [2] adapts the LBFGS algorithm to the special nonsmooth case in which $g(\cdot) \equiv \|\cdot\|_1$, and is popular for distributed optimization of ℓ_1-regularized ERM problems. Extension of this approach to other nonsmooth g is not well understood, and the convergence guarantees are only asymptotic, rather than global.

To the best of our knowledge, for ERMs with *general* nonsmooth regularizers in the instance-wise storage setting, proximal-gradient-like methods [17, 23] are the only practical distributed optimization algorithms with convergence guarantees. Since these methods barely use the Hessian information of the smooth part (if at all), we suspect that proper utilization of second-order information has the potential to improve convergence speed and therefore communication efficiency dramatically. We thus propose a practical distributed inexact variable-metric algorithm for general (1) which uses gradients and which updates information from previous iterations to estimate curvature of the smooth part f in a communication-efficient manner. We describe construction of this estimate and solution of the corresponding subproblem. We also provide convergence rate guarantees, which also bound communication complexity. These rates improve on existing distributed methods, even those tailor-made for specific regularizers.

Our algorithm leverages the more general framework provided in Lee and Wright [9], and our major contribution in this work is to describe how the main steps of the framework can be implemented efficiently in a distributed environment. Our approach has both good communication and computational complexity, unlike certain approaches that focus only on communication at the expense of computation (and ultimately overall time). We believe that this work is the first to propose, analyze, and implement a practically feasible distributed optimization method for solving (1) with general nonsmooth regularizer g under the instance-wise storage setting.

Our algorithm and implementation details are given in Section 2. Communication complexity and the effect of the subproblem solution inexactness are analyzed in Section 3. Section 4 discusses related works, and empirical comparisons are conducted in Section 5. Concluding observations appear in Section 6.

Notation

We use the following notation.

- $f(X^T\boldsymbol{w})$ is abbreviated as $\tilde{f}(\boldsymbol{w})$.
- $\|\cdot\|$ denotes the 2-norm, both for vectors and for matrices.
- Given any symmetric positive semi-definite matrix $H \in \mathbb{R}^{d\times d}$ and any vector $\boldsymbol{p} \in \mathbb{R}^d$, $\|\boldsymbol{p}\|_H$ denotes the semi-norm $\sqrt{\boldsymbol{p}^T H \boldsymbol{p}}$.

2 ALGORITHM

At each iteration of our algorithm for optimizing (1), we construct a subproblem that consists of a quadratic approximation of $\tilde{f}$ added to the original regularizer g. Specifically, given the current iterate $\boldsymbol{w}$, we choose a positive semi-definite H and define

$$Q_H(\boldsymbol{p};\boldsymbol{w}) \coloneqq \nabla\tilde{f}(\boldsymbol{w})^T\boldsymbol{p} + \frac{1}{2}\|\boldsymbol{p}\|_H^2 + g(\boldsymbol{w}+\boldsymbol{p}) - g(\boldsymbol{w}),$$

the update direction is obtained by approximately solving

$$\min_{\boldsymbol{p}\in\mathbb{R}^d} \quad Q_H(\boldsymbol{p};\boldsymbol{w}). \tag{2}$$

A line search procedure determines a suitable stepsize α, and we perform the update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{p}$.

We now discuss the following issues in the distributed setting: communication cost, the computation of $\nabla\tilde{f}$, the choice and construction of H, procedures for solving (2), and the line search procedure. In our description, we sometimes need to split some n-dimensional vectors over the machines, in accordance with the following disjoint partition $J_1,\ldots,J_k$ of $\{1,\ldots,d\}$:

$$J_i \cap J_k = \phi, \forall i \neq k, \quad \cup_{i=1}^K J_i = \{1,\ldots,d\}.$$

2.1 Communication Cost

For the ease of description, we assume the *allreduce* model of MPI [15], but it is also straightforward to extend the framework to a master-worker platform. Under this model, all machines simultaneously fulfill master and worker roles, and any data transmitted is broadcast to all machines. This can be considered as equivalent to conducting one map-reduce operation and then broadcasting the result to all nodes. The communication cost for the allreduce operation on a d-dimensional vector under this model is

$$\log(K)\, T_{\text{initial}} + dT_{\text{byte}}, \tag{3}$$

where T_{initial} is the latency to establish connection between machines, and T_{byte} is the per byte transmission time (see, for example, Chan et al. [5, Section 6.3]).

The first term in (3) also explains why batch methods are preferable. Even if methods that frequently update the iterates communicate the same amount of bytes, it takes more rounds of communication to transmit the information,

and the overhead of $\log(K)T_{\text{initial}}$ incurred at every round of communication makes this cost dominant, especially when K is large.

In subsequent discussion, when an allreduce operation is performed on a vector of dimension $O(d)$, we simply say that $O(d)$ communication is conducted. We omit the latency term since batch methods like ours tend to have only a small constant number of rounds of communication per iteration. By contrast, non-batch methods such as CD or stochastic gradient require $O(d)$ or $O(n)$ rounds of communication per epoch and therefore face much more significant latency issues.

2.2 Computing $\nabla \tilde{f}$

The gradient of $\tilde{f}$ has the form

$$\nabla \tilde{f}(\boldsymbol{w}) = X \nabla f(X^T \boldsymbol{w}) = \sum_{k=1}^{K} \left(X_k \nabla f_k(X_k^T \boldsymbol{w}) \right). \quad (4)$$

We see that, except for the sum over k, the computation can be conducted locally provided $\boldsymbol{w}$ is available to all machines. Our algorithm maintains $X_k^T \boldsymbol{w}$ on the kth machine throughout, and the most costly steps are the matrix-vector multiplications between X_k and $\nabla f_k(X_k^T \boldsymbol{w})$, and $X^T \boldsymbol{w}$. The local d-dimensional partial gradients are then aggregated through an allreduce operation.

2.3 Constructing a good H efficiently

We use the Hessian approximation constructed by the LBFGS algorithm [13], and propose a way to maintain and utilize it efficiently in a distributed setting. Using the compact representation in Byrd et al. [4], given a prespecified integer $m > 0$, at the tth iteration for $t > 0$, let $\tilde{m} \coloneqq \min(m, t)$, and define

$$\boldsymbol{s}_i \coloneqq \boldsymbol{w}_{i+1} - \boldsymbol{w}_i, \quad \boldsymbol{y}_i \coloneqq \nabla \tilde{f}(\boldsymbol{w}_{i+1}) - \nabla \tilde{f}(\boldsymbol{w}_i), \quad \forall i.$$

The LBFGS Hessian approximation matrix is

$$H_t = \gamma_t I - U_t M_t^{-1} U_t^T, \quad (5)$$

where

$$U_t \coloneqq [\gamma_t S_t, Y_t], \quad M_t \coloneqq \begin{bmatrix} \gamma_t S_t^T S_t, & L_t \\ L_t^T & -D_t \end{bmatrix}, \quad (6a)$$

$$\gamma_t \coloneqq \frac{\boldsymbol{s}_{t-1}^T \boldsymbol{s}_{t-1}}{\boldsymbol{s}_{t-1}^T y_{t-1}}, \quad (6b)$$

and

$$S_t \coloneqq [\boldsymbol{s}_{t-\tilde{m}}, \boldsymbol{s}_{t-\tilde{m}+1}, \ldots, \boldsymbol{s}_{t-1}], \quad (7a)$$

$$Y_t \coloneqq [\boldsymbol{y}_{t-\tilde{m}}, \boldsymbol{y}_{t-\tilde{m}+1}, \ldots, \boldsymbol{y}_{t-1}], \quad (7b)$$

$$D_t \coloneqq \operatorname{diag}\left(\boldsymbol{s}_{t-\tilde{m}}^T \boldsymbol{y}_{t-\tilde{m}}, \ldots, \boldsymbol{s}_{t-1}^T \boldsymbol{y}_{t-1} \right), \quad (7c)$$

$$(L_t)_{i,j} \coloneqq \begin{cases} \boldsymbol{s}_{t-m-1+i}^T \boldsymbol{y}_{t-m-1+j}, & \text{if } i > j, \\ 0, & \text{otherwise.} \end{cases} \quad (7d)$$

At the first iteration where no $\boldsymbol{s}_i$ and $\boldsymbol{y}_i$ are available, we set $H_0 \coloneqq a_0 I$ for some positive scalar a_0. When f is twice-differentiable and convex, we use

$$a_0 \coloneqq \frac{\|\nabla f(\boldsymbol{w}_0)\|^2_{\nabla^2 f(\boldsymbol{w}_0)}}{\|\nabla f(\boldsymbol{w}_0)\|^2}. \quad (8)$$

If f is not strongly convex, it is possible that (5) is only positive semi-definite. In this case, we follow Li and Fukushima [11], taking the m update pairs to be the most recent m iterations for which the inequality

$$\boldsymbol{s}_i^T \boldsymbol{y}_i \geq \delta \boldsymbol{s}_i^T \boldsymbol{s}_i \quad (9)$$

is satisfied, for some predefined $\delta > 0$. It can be shown that this safeguard ensures that H_t are always positive definite and the eigenvalues are bounded within a positive range (see, for example, the appendix of Lee and Wright [8]).

No additional communication is required to compute H_t. The gradients at all previous iterations have been shared with all machines through the *allreduce* operation, and the iterates $\boldsymbol{w}_t$ are also available on each machine, as they are needed to compute the local gradient. Thus the information needed to form H_t is available locally on each machine.

We now consider the costs associated with the matrix M_t. In practice, m is usually much smaller than d, so the $O(m^3)$ cost of inverting the matrix directly is insignificant compared to the cost of the other steps. However, if d is large, the computation of the inner products $\boldsymbol{s}_i^T \boldsymbol{y}_j$ and $\boldsymbol{s}_i^T \boldsymbol{s}_j$ can be expensive. We can significantly reduce this cost by computing and maintaining the inner products in parallel and assembling the results with $O(m)$ communication cost. At the tth iteration, given the new $\boldsymbol{s}_{t-1}$, we compute its inner products with both S_t and Y_t in parallel via the summations

$$\sum_{k=1}^{K} \left((S_t)_{J_k,:}^T (\boldsymbol{s}_{t-1})_{J_k} \right), \quad \sum_{k=1}^{K} \left((Y_t)_{J_k,:}^T (\boldsymbol{s}_{t-1})_{J_k} \right),$$

requiring $O(m)$ communication of the partial sums on each machine. We keep these results until $\boldsymbol{s}_{t-1}$ and $\boldsymbol{y}_{t-1}$ are discarded, so that at each iteration, only $2m$ (not $O(m^2)$) inner products are computed.

2.4 Solving the Subproblem

The approximate Hessian H_t is generally not diagonal, so there is no easy closed-form solution to (2). We will instead use iterative algorithms to obtain an approximate solution to this subproblem. In single-core environments, coordinate descent (CD) is one of the most efficient approaches for solving (2) [18, 25, 27]. Since the subproblem (2) is formed locally on all machines, a local CD process can be applied when g is separable and d is not too large. The alternative approach of applying proximal-gradient methods to (2) may be more efficient in distributed settings, since they can be parallelized with little communication cost for large d, and can be applied to larger classes of regularizers g.

The fastest proximal-gradient-type methods are accelerated gradient (AG) [17] and SpaRSA [23]. SpaRSA is a basic proximal-gradient method with spectral initialization of the

parameter in the prox term. SpaRSA has a few key advantages over AG despite its weaker theoretical convergence rate guarantees. It tends to be faster in the early iterations of the algorithm [24], thus possibly yielding a solution of acceptable accuracy in fewer iterations than AG. It is also a descent method, reducing the objective Q_H at every iteration, which ensures that the solution returned is at least as good as the original guess $\boldsymbol{p} = 0$

In the rest of this subsection, we will describe a distributed implementation of SpaRSA for (2), with H as defined in (5). To distinguish between the iterations of our main algorithm (i.e. the entire process required to update $\boldsymbol{w}$ a single time) and the iterations of SpaRSA, we will refer to them by *main iterations* and *SpaRSA iterations* respectively.

Since H and $\boldsymbol{w}$ are fixed in this subsection, we will write $Q_H(\cdot;\boldsymbol{w})$ simply as $Q(\cdot)$. We denote the ith iterate of the SpaRSA algorithm as $\boldsymbol{p}^{(i)}$, and we initialize $\boldsymbol{p}^{(0)} \equiv 0$. We denote the smooth part of Q_H by $\hat{f}(\boldsymbol{p})$, and the nonsmooth $g(\boldsymbol{w}+\boldsymbol{p})$ by $\hat{g}(\boldsymbol{p})$. At the ith iteration of SpaRSA, we define

$$\mathbf{u}_{\psi_i}^{(i)} \coloneqq \boldsymbol{p}^{(i)} - \frac{\nabla \hat{f}(\boldsymbol{p}^{(i)})}{\psi_i}, \tag{10}$$

and solve the following subproblem:

$$\boldsymbol{p}^{(i+1)} = \arg\min_{\boldsymbol{p}} \frac{1}{2}\left\|\boldsymbol{p} - \mathbf{u}_{\psi_i}^{(i)}\right\|^2 + \frac{\hat{g}(\boldsymbol{p})}{\psi_i}, \tag{11}$$

where ψ_i is defined by the following "spectral" formula:

$$\psi_i = \frac{\left(\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(i-1)}\right)^T \left(\nabla \hat{f}(\boldsymbol{p}^{(i)}) - \nabla \hat{f}(\boldsymbol{p}^{(i-1)})\right)}{\left\|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(i-1)}\right\|^2}. \tag{12}$$

When $i = 0$, we use a pre-assigned value for ψ_0 instead. (In our LBFGS choice for H_t, we use the value of γ_t from (6b) as the initial estimate of ψ_0.) The exact minimizer of (11) can be difficult to compute for general regularizers g. However, approximate solutions of (11) suffice to provide a convergence rate guarantee for solving (2) [6, 9, 18, 19]. Since it is known (see [11]) that the eigenvalues of H are upper- and lower-bounded in a positive range after the safeguard (9) is applied, we can guarantee that this initialization of ψ_i is bounded within a positive range; see Section 3. The initial value of ψ_i defined in (12) is increased successively by a chosen constant factor $\beta > 1$, and $\boldsymbol{p}^{(i+1)}$ is recalculated from (11), until the following sufficient decrease criterion is satisfied:

$$Q\left(\boldsymbol{p}^{(i+1)}\right) \le Q\left(\boldsymbol{p}^{(i)}\right) - \frac{\psi_i \sigma_0}{2}\left\|\boldsymbol{p}^{(i+1)} - \boldsymbol{p}^{(i)}\right\|^2, \tag{13}$$

for some specified $\sigma_0 \in (0,1)$. Note that the evaluation of $Q(\boldsymbol{p})$ needed in (13) can be done efficiently through a parallel computation of $(\nabla \hat{f}(\boldsymbol{p}) + \nabla \tilde{f}(\boldsymbol{w}))^T \boldsymbol{p}/2$ plus the $\hat{g}(\boldsymbol{p})$ term. From the boundedness of H, one can easily prove that (13) is satisfied after a finite number of increases of ψ_i, as we will show in Section 3. In our algorithm, SpaRSA runs until either a fixed number of iterations is reached, or when some certain inner stopping condition for optimizing (2) is satisfied.

For general H, the computational bottleneck of $\nabla \hat{f}$ would take $O(d^2)$ operations to compute the $H\boldsymbol{p}^{(i)}$ term. However, for our LBFGS choice of H_k, this cost is reduced to $O(md + m^2)$ by utilizing the matrix structure, as shown in the following formula:

$$\nabla \hat{f}\left(\boldsymbol{p}\right) = \nabla \tilde{f}\left(\boldsymbol{w}\right) + H\boldsymbol{p} = \nabla \tilde{f}(\boldsymbol{w}) + \gamma \boldsymbol{p} - U_t\left(M_t^{-1}\left(U_t^T \boldsymbol{p}\right)\right). \tag{14}$$

The computation of (14) can be parallelized, by first parallelizing computation of the inner product $U_t^T \boldsymbol{p}^{(i)}$ via the formula

$$\sum_{k=1}^{K} (U_t)_{J_k,:}^T \boldsymbol{p}_{J_k}^{(i)}$$

with $O(m)$ communication. (We implement the parallel inner products as described in Section 2.3.) We either construct the whole vector $\mathbf{u}$ in (10) on all machines, or let each machine compute a subvector of $\mathbf{u}$ in (10). The former scheme is most suitable when g is non-separable, but the latter has a lower computational burden per machine, in cases for which it is feasible to apply. We describe the latter scheme in more detail. The kth machine locally computes $\boldsymbol{p}_{J_k}^{(i)}$ without communicating the whole vector. Then at each iteration of SpaRSA, partial inner products between $(U_t)_{J_k,:}$ and $\boldsymbol{p}_{J_k}^{(i)}$ can be computed locally, and the results are assembled with one $O(m)$ communication. This technique also suggests a spatial advantage of our method: The rows of S_t and Y_t can be stored in a distributed manner consistent with the subvector partition. This approach incurs $O(m)$ communication cost per SpaRSA iteration, with the computational cost reduced from $O(md)$ to $O(md/K)$ per machine. Since both the $O(m)$ communication cost and the $O(md/K)$ computational cost are inexpensive when m is small, in comparison to the computation of $\nabla \tilde{f}$, one can afford to conduct multiple iterations of SpaRSA at every main iteration. Note that the latency incurred at every communication as discussed in (3) can be capped by setting a maximum iteration limit for SpaRSA. Finally, after the SpaRSA procedure terminates, all machines conduct one $O(d)$ communication to gather the update step $\boldsymbol{p}$.

The distributed implementation of SpaRSA for solving (2) is summarized in Algorithm 1.

2.5 Line Search

After obtaining an update direction $\boldsymbol{p}^k$ by approximately minimizing $Q_{H_k}(\cdot;\boldsymbol{w}_k)$, a line search procedure is usually needed to find a step size α_k that ensures sufficient decrease in the objective value. We follow Tseng and Yun [21] by using a modified-Armijo-type backtracking line search to find a suitable step size α. Given the current iterate $\boldsymbol{w}$, the update direction $\boldsymbol{p}$, and parameters $\sigma_1, \theta \in (0,1)$, we set

$$\Delta \coloneqq \nabla \tilde{f}\left(\boldsymbol{w}\right)^T \boldsymbol{p} + g\left(\boldsymbol{w}+\boldsymbol{p}\right) - g\left(\boldsymbol{w}\right) \tag{15}$$

and pick the step size as the largest of $\theta^0, \theta^1, \ldots$ satisfying

$$F\left(\boldsymbol{w} + \alpha \boldsymbol{p}\right) \le F\left(\boldsymbol{w}\right) + \alpha \sigma_1 \Delta. \tag{16}$$

Algorithm 1: Distributed SpaRSA for solving (2) with LBFGS quadratic approximation on machine k

Given $\beta > 1$, $\sigma_0 \in (0,1)$, M_t^{-1}, U_t, and γ_t;
Set $\boldsymbol{p}_{J_k}^{(0)} \leftarrow 0$;
for $i = 0, 1, 2, \ldots$ **do**
  if $i = 0$ **then**
    $\psi = \gamma_t$;
  else
    Compute ψ in (12) through

$$\sum_{j=1}^{K} \left(\boldsymbol{p}_{J_j}^{(i)} - \boldsymbol{p}_{J_j}^{(i-1)}\right)^T \left(\nabla_{J_j} \hat{f}\left(\boldsymbol{p}^{(i)}\right) - \nabla_{J_j} \hat{f}\left(\boldsymbol{p}^{(i-1)}\right)\right),$$

    and

$$\sum_{j=1}^{K} \left\|\boldsymbol{p}_{J_j}^{(i)} - \boldsymbol{p}_{J_j}^{(i-1)}\right\|^2;$$

    ▷ $O(1)$ `comm.`
  end if
  Obtain ▷ $O(m)$ `comm.`

$$U_t^T \boldsymbol{p}^{(i)} = \sum_{j=1}^{K} (U_t)_{J_j,:}^T \boldsymbol{p}_{J_j}^{(i)};$$

  Compute

$$\nabla_{J_k} \hat{f}\left(\boldsymbol{p}^{(i)}\right) = \nabla_{J_k} \tilde{f}(\boldsymbol{w}) + \gamma \boldsymbol{p}_{J_k}^{(i)} - (U_t)_{J_k,:} \left(M_t^{-1}\left(U_t^T \boldsymbol{p}^{(i)}\right)\right)$$

  by (14);
  while TRUE **do**
    Solve (11) on coordinates indexed by J_k to obtain $\boldsymbol{p}_{J_k}$;
    if (13) holds ▷ $O(1)$ `comm.` **then**
      $\boldsymbol{p}_{J_k}^{(i+1)} \leftarrow \boldsymbol{p}_{J_k}$; $\psi_i \leftarrow \psi$;
      Break;
    end if
    $\psi \leftarrow \beta\psi$;
    Re-solve (11) with the new ψ to obtain a new $\boldsymbol{p}_{J_k}$;
  end while
  Break if some stopping condition is met;
end for
Gather the final solution $\boldsymbol{p}$ ▷ $O(d)$ `comm.`

The computation of Δ can again be done in a distributed manner. First, $X_k^T \boldsymbol{p}$ can be computed locally on each machine, then the first term in (15) is obtained by sending a scalar over the network. When g is block-separable, its computation can also be distributed across machines. The vector $X_k^T \boldsymbol{p}$ is then used to compute the left-hand side of (16) for arbitrary values of α. Writing $X_k^T(\boldsymbol{w} + \alpha\boldsymbol{p}) = \left(X_k^T \boldsymbol{w}\right) + \alpha\left(X_k^T \boldsymbol{p}\right)$, we see that once $X_k^T \boldsymbol{w}$ and $X_k^T \boldsymbol{p}$ are known, we can evaluate $X_k^T(\boldsymbol{w} + \alpha\boldsymbol{p})$ via an "axpy" operation (weighted sum of two vectors). Because H_t defined in (5) attempts to approximate the real Hessian, the unit step $\alpha = 1$ frequently satisfies (16), so we use the value 1 as the initial guess. Aside from the communication needed to compute the summation of the f_k terms in the evaluation of F, the only other communication needed is to share the update direction $\boldsymbol{p}$ if (2) was solved in a distributed manner. Thus, two rounds of $O(d)$ communication are incurred per main iteration. Otherwise, if each machine solves the same subproblem (2) locally, then only one round of $O(d)$ communication is required.

Algorithm 2: A distributed proximal variable-metric LBFGS method with line search for (1)

Given $\theta, \sigma_1 \in (0,1)$, $\delta > 0$, an initial point $\boldsymbol{w} = \boldsymbol{w}_0$, distributed $X = [X_1, \ldots, X_K]$;
for Machines $k = 1, \ldots, K$ in parallel **do**
  Compute $X_k^T \boldsymbol{w}$ and $f_k(X_k^T \boldsymbol{w})$;
  $H \leftarrow aI$ for some $a > 0$ (use (8) if possible);
  Obtain $F(\boldsymbol{w})$; ▷ $O(1)$ `comm.`
  for $t = 0, 1, 2, \ldots$ **do**
    Compute $\nabla \tilde{f}(\boldsymbol{w})$ through (4); ▷ $O(d)$ `comm.`
    if $t \neq 0$ and (9) holds for $(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1})$ **then**
      Update U, M, and γ by (6)-(7); ▷ $O(m)$ `comm.`
      Construct a new H from (5);
    end if
    if $H = aI$ **then**
      Solve (2) directly to obtain $\boldsymbol{p}$;
    else
      Solve (2) using Algorithm 1 either in a distributed manner or locally to obtain $\boldsymbol{p}$;
    end if
    Compute $X_k^T \boldsymbol{p}$;
    Compute Δ defined in (15); ▷ $O(1)$ `comm.`
    for $i = 0, 1, \ldots$ **do**
      $\alpha = \theta^i$;
      Compute $(X_k^T \boldsymbol{w}) + \alpha(X_k^T \boldsymbol{p})$;
      Compute $F(\boldsymbol{w} + \alpha\boldsymbol{p})$; ▷ $O(1)$ `comm.`
      if $F(\boldsymbol{w} + \alpha\boldsymbol{p}) \leq F(\boldsymbol{w}) + \sigma_1\alpha\Delta$ **then**
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{p}$, $F(\boldsymbol{w}) \leftarrow F(\boldsymbol{w} + \alpha\boldsymbol{p})$;
        $X_k^T \boldsymbol{w} \leftarrow X_k^T \boldsymbol{w} + \alpha X_k^T \boldsymbol{p}$;
        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}$;
        $\boldsymbol{s}_t \leftarrow \boldsymbol{w}_{t+1} - \boldsymbol{w}_t$, $\boldsymbol{y}_t \leftarrow \nabla \tilde{f}(\boldsymbol{w}_{t+1}) - \nabla \tilde{f}(\boldsymbol{w}_t)$;
        Break;
      end if
    end for
  end for
end for

Our distributed algorithm for (1) is summarized in Algorithm 2.

2.6 Cost Analysis

We now summarize the costs of our algorithm. For the distributed version of Algorithm 1, each iteration costs

$$O\left(\frac{d}{K} + \frac{md}{K} + m^2\right) = O\left(\frac{md}{K} + m^2\right) \tag{17}$$

in computation and

$$O\left(m + 1 \times \text{number of times (13) is evaluated}\right)$$

in communication. In the next section, we will show that (13) is accepted in a constant number of times and thus the overall communication cost is $O(m)$.

For Algorithm 2, the computational cost per iteration is

$$O\left(\frac{\#\text{nnz}}{K}+\frac{n}{K}+d+\frac{md}{K}+\frac{d}{K}\right)=O\left(\frac{\#\text{nnz}}{K}+d+\frac{md}{K}\right), \tag{18}$$

where #nnz is the number of nonzero elements in X, and the communication cost is

$$O\left(1+m+d\right)=O\left(d\right).$$

We note that the costs of Algorithm 1 are dominated by those of Algorithm 2 if a fixed number of SpaRSA iterations is conducted every main iteration.

3 COMMUNICATION COMPLEXITY

The use of an iterative solver for the subproblem (2) generally results in an inexact solution. We first show that running SpaRSA for any fixed number of iterations guarantees a step $\boldsymbol{p}$ whose accuracy is sufficient to prove overall convergence.

LEMMA 3.1. *Using H_t as defined in (5) with the safeguard mechanism (9) in (2), we have the following.*

(1) There exist constants $c_1 \geq c_2 > 0$ such that $c_1 I \succeq H_t \succeq c_2 I$ for all main iterations. Moreover, $\|X^T X\| L \geq \gamma_t \geq \delta$ for all $t > 0$.

(2) The initial estimate of ψ_i at every SpaRSA iteration is bounded within the range of $[\min\{c_2, \delta\}, \max\{c_1, \|X^T X\| L\}]$, and the final accepted value ψ_i is upper-bounded.

(3) SpaRSA is globally Q-linear convergent in solving (2). Therefore, there exists $\eta \in [0, 1)$ such that if we run at least S iterations of SpaRSA for all main iterations for any $S > 0$, the approximate solution $\boldsymbol{p}$ satisfies

$$-\eta^S Q^* = \eta^S \left(Q\left(0\right) - Q^*\right) \geq Q\left(\boldsymbol{p}\right) - Q^*, \tag{19}$$

where Q^* is the optimal objective of (2).

Lemma 3.1 establishes how the number of iterations of SpaRSA affects the inexactness of the subproblem solution. Given this measure, we can leverage the results developed in Lee and Wright [9] to obtain iteration complexity guarantees for our algorithm. Since in our algorithm, communication complexity scales linearly with iteration complexity, this guarantee provides a bound on the amount of communication. In particular, our method communicates $O(d + mS)$ bytes per iteration (where S is the number of SpaRSA iterations used, as in Lemma 3.1) and the second term can usually be ignored for small m.

We show next that the step size generated by our line search procedure in Algorithm 2 is lower bounded by a positive value.

LEMMA 3.2. *If SpaRSA is run at least S iterations in solving (2), the corresponding Δ defined in (15) satisfies*

$$\Delta \leq -\frac{c_2 \left\|d\right\|^2}{\left(1+\eta^{S/2}\right)}. \tag{20}$$

Moreover, the backtracking subroutine in Algorithm 2 terminates in finite steps and produces a step size

$$\alpha \geq \bar{\alpha} \geq \min\left\{1, \frac{2\theta\left(1-\sigma_1\right)c_2}{\|X^T X\| L\left(1+\eta^{S/2}\right)}\right\}. \tag{21}$$

This result is just a worst-case guarantee; in practice we often observe that the line search procedure terminates with $\alpha = 1$ for our choice of H, as we see in our experiments.

Next, we analyze communication complexity of Algorithm 2.

THEOREM 3.3. *If we apply Algorithm 2 to solve (1), and Algorithm 1 is run for S iterations at each main iteration, then the following claims hold.*

- *Suppose that the following variant of strong convexity holds: There exists $\mu > 0$ such that for any $\boldsymbol{w}$ and any $a \in [0, 1]$, we have*

$$\begin{aligned} &F\left(a\boldsymbol{w} + (1-a)\, P_\Omega\left(\boldsymbol{w}\right)\right) \\ &\leq aF\left(\boldsymbol{w}\right) + (1-a)\, F^* - \frac{\mu a\left(1-a\right)}{2}\left\|\boldsymbol{w} - P_\Omega\left(\boldsymbol{w}\right)\right\|^2, \end{aligned} \tag{22}$$

  where F^* is the optimal objective value of (1), Ω is the solution set, and P_Ω is the projection onto this set. Then Algorithm 2 converges globally at a Q-linear rate. That is,

$$\frac{F\left(\boldsymbol{w}_{t+1}\right) - F^*}{F\left(\boldsymbol{w}_t\right) - F^*} \leq 1 - \frac{\bar{\alpha}\sigma_1\left(1-\eta^S\right)\mu}{\mu + c_1}, \quad \forall t.$$

  Therefore, to get an approximate solution of (1) that is ϵ-accurate in the sense of objective value, we need to perform at most

$$O\left(\frac{\mu + c_1}{\mu\sigma_1\bar{\alpha}\left(1-\eta^S\right)}\log\frac{1}{\epsilon}\right) \tag{23}$$

  rounds of $O(d)$ communication.

- *When F is convex, and the level set defined by $\boldsymbol{w}_0$ is bounded, define*

$$R_0 \coloneqq \sup_{\boldsymbol{w}: F(\boldsymbol{w}) \leq F(\boldsymbol{w}_0)} \left\|\boldsymbol{w} - P_\Omega(\boldsymbol{w})\right\|.$$

  Then we obtain the following expressions for rate of convergence of the objective value.

  (1) When $F(\boldsymbol{w}_t) - F^* \geq c_1 R_0^2$,

$$\frac{F(\boldsymbol{w}_{t+1}) - F^*}{F(\boldsymbol{w}_t) - F^*} \leq 1 - \frac{\left(1-\eta^S\right)\sigma_1\bar{\alpha}}{2}.$$

  (2) Otherwise, we have globally for all t that

$$\frac{F\left(\boldsymbol{w}_t\right) - F^*}{2} \leq \frac{c_1 R_0^2 + F\left(\boldsymbol{w}_0\right) - F^*}{\sigma_1 t(1-\eta^S)\bar{\alpha}}.$$

  This implies a communication complexity of

$$\begin{cases} O\left(\frac{2}{\left(1-\eta^S\right)\sigma_1\bar{\alpha}}\log\frac{1}{\epsilon}\right) & \text{if } \epsilon \geq c_1 R_0^2, \\ \frac{2\left(c_1 R_0^2 + F(\boldsymbol{w}_0) - F^*\right)}{\sigma_1\left(1-\eta^S\right)\bar{\alpha}\epsilon} & \text{else.} \end{cases}$$

- *If F is non-convex, the norm of the proximal gradient steps*

$$G_t \coloneqq \arg\min_{\boldsymbol{p}} \nabla f\left(\boldsymbol{w}_t\right)^T \boldsymbol{p} + \frac{\|\boldsymbol{p}\|^2}{2} + g\left(\boldsymbol{w}_t + \boldsymbol{p}\right)$$

converge to zero at a rate of $O(1/\sqrt{t})$ *in the following sense:*

$$\min_{0\le i\le t}\|G_i\|^2 \le \frac{F(\boldsymbol{w}_0)-F^*}{\gamma(t+1)}\frac{c_1^2\left(1+\frac{1}{c_2}+\sqrt{1-\frac{2}{c_1}+\frac{1}{c_2^2}}\right)^2}{2c_2\bar{\alpha}(1-\eta^S)}.$$

Note that it is known that the norm of G_t is zero if and only if $\boldsymbol{w}_t$ is a stationary point [9], so this measure serves as a first-order optimality condition.

Our computational experiments cover the case of F convex; exploration of the method on nonconvex F is left for future work.

4 RELATED WORKS

The framework of using (2) to generate update directions for optimizing (1) has been discussed in existing works with different choices of H, but always in the single-core setting. Lee et al. [10] focused on using $\nabla^2\tilde{f}$ as H, and proved local convergence results under certain additional assumptions. In their experiment, they used AG to solve (2). However, in distributed environments, using $\nabla^2\tilde{f}$ as H incurs an $O(d)$ communication cost per AG iteration in solving (2), because computation of the term $\nabla^2\tilde{f}(\boldsymbol{w})\boldsymbol{p} = X\nabla^2 f(X^T\boldsymbol{w})X^T\boldsymbol{p}$ requires one *allreduce* operation to calculate a weighted sum of the columns of X.

Scheinberg and Tang [18] and Ghanbari and Scheinberg [6] showed global convergence rate results for a method based on (2) with bounded H, and suggested using randomized coordinate descent to solve (2). In the experiments of these two works, they used the same choice of H as we do in this paper, with CD as the solver for (2), which is well suited to their single-machine setting. Aside from our extension to the distributed setting and the use of SpaRSA, the third major difference between their algorithm and ours is that they do not conduct line search on the step size. Instead, when the obtained solution with a unit step size does not result in sufficient objective value decrease, they add a scaled identity matrix to H and solve (2) again starting from $\boldsymbol{p}^{(0)} = 0$. The cost of repeatedly solving (2) from scratch can be high, which results in an algorithm with higher overall complexity. This potential inefficiency is exacerbated further by the inefficiency of coordinate descent in the distributed setting.

Our method can be considered as a special case of the algorithmic framework in Bonettini et al. [3], Lee and Wright [9], which both focus on analyzing the theoretical guarantees under various conditions. In the experiments of Bonettini et al. [3], H is obtained from the diagonal entries of $\nabla^2\tilde{f}$, making the subproblem (2) easy to solve, but this simplification does not take full advantage of curvature information. Although our theoretical convergence analysis follows directly from Lee and Wright [9], that paper does not provide details of experimental results or implementation, and its analysis focuses on general H rather than the LBFGS choice we use here.

Some methods consider solving (1) in a distributed environment where X is partitioned feature-wise (i.e. along rows instead of columns). There are two potential disadvantages of this approach. First, new data points can easily be assigned to one of the machines in our approach, whereas in the feature-wise approach, the features of all new points would need to be distributed around the machines. Second, local curvature information is obtained, so the update direction can be poor if the data is distributed nonuniformly across features. (Data is more likely to be distributed evenly across instances than across features.) In the extreme case in which each machine contains only one row of X, only the diagonal entries of the Hessian can be obtained locally, so the method reduces to a scaled version of proximal gradient.

Table 1: Data statistics.

Data set	n	d	#nonzeros
news	19,996	1,355,191	9,097,916
epsilon	400,000	2,000	800,000,000
webspam	350,000	16,609,143	1,304,697,446
avazu-site	25,832,830	999,962	387,492,144

5 NUMERICAL EXPERIMENTS

We investigate the empirical performance of Algorithm 2 in solving ℓ_1-regularized logistic regression problems. The code used in our experiment is available at http://github.com/leepei/dplbfgs/. Given training data points $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ for $i = 1, \ldots, n$, the objective function is

$$F(\boldsymbol{w}) = C\sum_{i=1}^{n}\log\left(1+e^{-y_i\boldsymbol{x}_i^T\boldsymbol{w}}\right) + \|\boldsymbol{w}\|_1, \qquad (24)$$

where $C > 0$ is a parameter prespecified to trade-off between the loss term and the regularization term. We fix C to 1 for simplicity in our experiments. We consider the publicly available binary classification data sets listed in Table 1[1], and partitioned the instances evenly across machines.

The parameters of our algorithm were set as follows: $\theta = 0.5$, $\beta = 2$, $\sigma_0 = 10^{-2}$, $\sigma_1 = 10^{-4}$, $m = 10$, $\delta = 10^{-10}$. The parameters in SpaRSA follow the setting in [23], θ is set to halve the step size each time, the value of σ_0 follows the default experimental setting of [7], δ is set to a small enough number, and $m = 10$ is a common choice for LBFGS.

We ran our experiments on a local cluster of 16 machines running MPICH2, and all algorithms are implemented in C/C++. The inversion of M defined in (6) is performed through LAPACK [1]. The comparison criteria are the relative objective error $(F(\boldsymbol{w}) - F^*)/F^*$, versus either the amount communicated (divided by d) or the overall running time. The former criterion is useful in estimating the performance in environments in which communication cost is extremely high.

[1]Downloaded from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

Table 2: Different stopping conditions of SpaRSA as an approximate solver for (2). We show required amount of communication (divided by d) and running time (in seconds) to reach $F(\boldsymbol{w}_t) - F^* \leq 10^{-3}F^*$.

Data set	ϵ_1	Communication	Time
news20	10^{-1}	28	11
	10^{-2}	25	11
	10^{-3}	23	14
epsilon	10^{-1}	144	45
	10^{-2}	357	61
	10^{-3}	687	60
webspam	10^{-1}	452	3254
	10^{-2}	273	1814
	10^{-3}	249	1419

5.1 Effect of Inexactness in the Subproblem Solution

We first examine how the degree of inexactness of the approximate solution of subproblems (2) affects the convergence of the overall algorithm. Instead of treating SpaRSA as a steadily linearly converging algorithm, we take it as an algorithm that sometimes decreases the objective much faster than the worst-case guarantee, thus an adaptive stopping condition is used. In particular, we terminate Algorithm 1 when the norm of the current update step is smaller than ϵ_1 times that of the first update step, for some prespecified $\epsilon_1 > 0$. From the proof of Lemma 3.1, the norm of the update step bounds the value of $Q(\boldsymbol{p}) - Q^*$ both from above and from below, and thus serves as a good measure of the solution precision. In Table 2, we compare runs with the values $\epsilon_1 = 10^{-1}, 10^{-2}, 10^{-3}$. For the datasets news20 and webspam, it is as expected that tighter solution of (2) results in better updates and hence lower communication cost. This may not result in a longer convergence time. As for the dataset epsilon, which has a smaller data dimension d, the $O(m)$ communication cost per SpaRSA iteration for calculating $\nabla \tilde{f}$ is significant in comparison. In this case, setting a tighter stopping criteria for SpaRSA can result in higher communication cost and longer running time.

In Table 3, we show the distribution of the step sizes over the main iterations, for the same set of values of ϵ_1. As we discussed in Section 3, although the smallest α can be much smaller than one, the unit step is usually accepted. Therefore, although the worst-case communication complexity analysis is dominated by the smallest step encountered, the practical behavior is much better.

5.2 Comparison with Other Methods

Now we compare our method with two state-of-the-art distributed algorithms for (1). In addition to a proximal-gradient-type method that can be used to solve general (1) in distributed environments easily, we also include one solver specifically designed for ℓ_1-regularized problems in our comparison. These methods are:

Table 3: Step size distributions.

Data set	ϵ_1	percent of $\alpha = 1$	smallest α
news20	10^{-1}	95.5%	2^{-3}
	10^{-2}	95.5%	2^{-4}
	10^{-3}	95.5%	2^{-3}
epsilon	10^{-1}	96.8%	2^{-5}
	10^{-2}	93.4%	2^{-6}
	10^{-3}	91.2%	2^{-3}
webspam	10^{-1}	98.5%	2^{-3}
	10^{-2}	97.6%	2^{-2}
	10^{-3}	97.2%	2^{-2}

- DPLBFGS: our Distributed Proximal LBFGS approach. We fix $\epsilon_1 = 10^{-2}$ in this experiment.
- SPARSA [23]: the method described in Section 2.4, but applied directly to (1).
- OWLQN [2]: an orthant-wise quasi-Newton method specifically designed for ℓ_1-regularized problems. We fix $m = 10$ in the LBFGS approximation.

We implement all methods in C/C++ and MPI. Note that the AG method [17] can also be used, but its empirical performance has been shown to be similar to SpaRSA [24] and it requires strong convexity and Lipschitz parameters to be estimated, which induces an additional cost. A further examination on different values of m indicates that convergence speed of our method improves with larger m, while in OWLQN, larger m usually does not lead to better results. We use the same value of m for both methods and choose a value that favors OWLQN.

The results are provided in Figure 1. Our method is always the fastest in both criteria. For epsilon, our method is orders of magnitude faster, showing that correctly using the curvature information of the smooth part is indeed beneficial in reducing the communication complexity.

It is possible to include specific heuristics for ℓ_1-regularized problems, such as those applied in Yuan et al. [25], Zhong et al. [27], to further accelerate our method, and the exploration on this direction is an interesting topic for future work.

6 CONCLUSIONS

In this work, we propose a practical and communication-efficient distributed algorithm for solving general regularized nonsmooth ERM problems. Our algorithm enjoys fast performance both theoretically and empirically and can be applied to a wide range of ERM problems. It is possible to extend our approach for solving the distributed dual ERM problem with a strongly convex primal regularizer, and we expect our framework to outperform state of the art, which only uses block-diagonal parts of the Hessian that can be computed locally. These topics are left for future work.

ACKNOWLEDGEMENT

This work was supported by NSF Awards IIS-1447449, 1628384, 1634597, and 1740707; AFOSR Award FA9550-13-1-0138; and

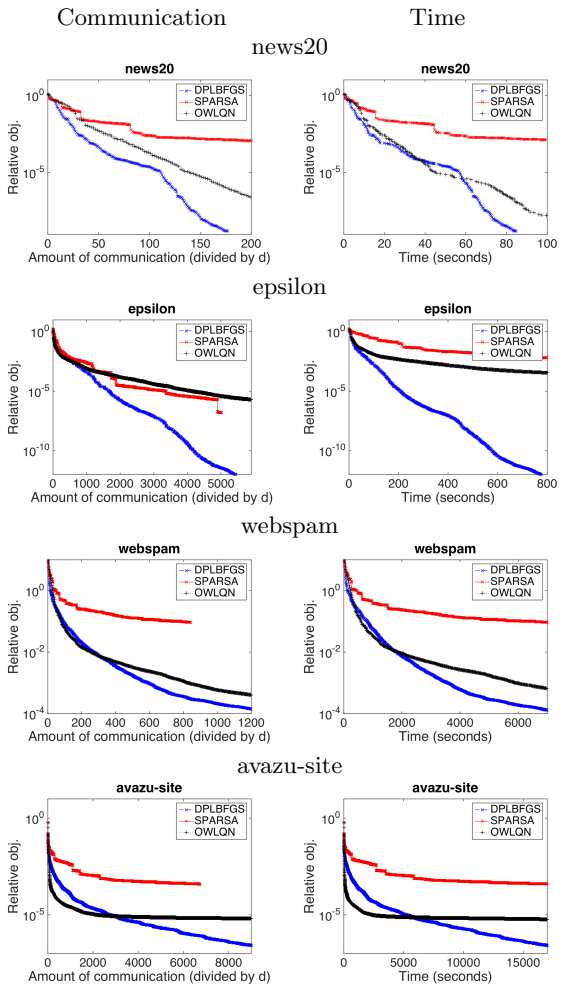

Figure 1: Comparison between different methods for ℓ_1-regularized logistic regression in terms of relative objective difference to the optimum. Left: communication (divided by d); right: running time (in seconds).

Subcontract 3F-30222 from Argonne National Laboratory. The authors thank En-Hsu Yen for fruitful discussions.

REFERENCES

[1] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. 1999. *LAPACK Users' guide*. SIAM.

[2] Galen Andrew and Jianfeng Gao. 2007. Scalable Training of L1-Regularized Log-Linear Models. In *International Conference on Machine Learning*.

[3] Silvia Bonettini, Ignace Loris, Federica Porta, and Marco Prato. 2016. Variable metric inexact line-search-based methods for nonsmooth optimization. *SIAM journal on optimization* 26, 2 (2016), 891–921.

[4] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. 1994. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming* 63, 1-3 (1994), 129–156.

[5] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert Van De Geijn. 2007. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 19, 13 (2007), 1749–1783.

[6] Hiva Ghanbari and Katya Scheinberg. 2016. *Proximal Quasi-Newton Methods for Convex Optimization*. Technical Report. arXiv:1607.03081.

[7] Ching-pei Lee, Po-Wei Wang, Weizhu Chen, and Chih-Jen Lin. 2017. Limited-memory Common-directions Method for Distributed Optimization and its Application on Empirical Risk Minimization. In *Proceedings of SIAM International Conference on Data Mining*.

[8] Ching-pei Lee and Stephen J. Wright. 2017. *Using Neural Networks to Detect Line Outages from PMU Data*. Technical Report. arXiv:1710.05916.

[9] Ching-pei Lee and Stephen J. Wright. 2018. *Inexact Successive Quadratic Approximation for Regularized Optimization*. Technical Report.

[10] Jason D. Lee, Yuekai Sun, and Michael A. Saunders. 2014. Proximal Newton-type methods for minimizing composite functions. *SIAM Journal on Optimization* 24, 3 (2014), 1420–1443.

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

[12] Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, and Chih-Jen Lin. 2014. Large-scale logistic regression and linear support vector machines using Spark. In *Proceedings of the IEEE International Conference on Big Data*. 519–528.

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

[14] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[15] Message Passing Interface Forum. 1994. MPI: a message-passing interface standard. *International Journal on Supercomputer Applications* 8, 3/4 (1994).

[16] Yu Nesterov. 1983. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady* 27 (1983), 372–376.

[17] Yu Nesterov. 2013. Gradient methods for minimizing composite functions. *Mathematical Programming* 140, 1 (2013), 125–161.

[18] Katya Scheinberg and Xiaocheng Tang. 2016. Practical inexact proximal quasi-Newton method with global complexity analysis. *Mathematical Programming* 160, 1-2 (2016), 495–529.

[19] Mark Schmidt, Nicolas Roux, and Francis Bach. 2011. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in neural information processing systems*. 1458–1466.

[20] Ohad Shamir, Nati Srebro, and Tong Zhang. 2014. Communication-efficient distributed optimization using an approximate Newton-type method. In *International conference on machine learning*.

[21] Paul Tseng and Sangwoon Yun. 2009. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming* 117, 1 (2009), 387–423.

[22] Po-Wei Wang, Ching-pei Lee, and Chih-Jen Lin. 2016. *The Common Directions Method for Regularized Empirical Loss Minimization*. Technical Report. National Taiwan University.

[23] Stephen J. Wright, Robert D. Nowak, and Mário A. T. Figueiredo. 2009. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing* 57, 7 (2009), 2479–2493.

[24] Junfeng Yang and Yin Zhang. 2011. Alternating direction algorithms for ℓ1-problems in compressive sensing. *SIAM journal on scientific computing* 33, 1 (2011), 250–278.

[25] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. 2012. An Improved GLMNET for L1-regularized Logistic Regression. *Journal of Machine Learning Research* 13 (2012), 1999–2030.

[26] Yuchen Zhang and Xiao Lin. 2015. DiSCO: Distributed Optimization for Self-Concordant Empirical Loss. In *International Conference on Machine Learning*.

[27] Kai Zhong, Ian En-Hsu Yen, Inderjit S. Dhillon, and Pradeep K. Ravikumar. 2014. Proximal quasi-Newton for computationally intensive l1-regularized M-estimators. In *Advances in Neural*

Information Processing Systems.
[28] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. 2015. Distributed Newton method for regularized logistic regression. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD).*

A PROOFS

We provide proof of Lemma 3.1 in this section. The rest of Section 3 directly follows the results in Lee and Wright [9] by noting that $\nabla\tilde{f}(\boldsymbol{w})$ is $\|X^TX\|L$-Lipschitz continuous, and are therefore omitted.

Proof of Lemma 3.1. We prove the three results separately.

(1) The boundedness of H_t directly follow from the results in Li and Fukushima [11]. A more detailed proof can be found in, for example, Lee and Wright [8, Appendix E]. The lower bound of γ_t is directly through (9), and the upper bound is from the Lipschitz continuity of $\nabla\tilde{f}$.

(2) By directly expanding $\nabla\hat{f}$, we have that for any $\boldsymbol{p}_1, \boldsymbol{p}_2$,

$$\begin{aligned}\nabla\hat{f}(\boldsymbol{p}_1) - \nabla\hat{f}(\boldsymbol{p}_2) &= \nabla\tilde{f}(\boldsymbol{w}) + H\boldsymbol{p}_1 - \left(\nabla\tilde{f}(\boldsymbol{w}) + H\boldsymbol{p}_2\right)\\ &= H(\boldsymbol{p}_1 - \boldsymbol{p}_2).\end{aligned}$$

Therefore, we have

$$\frac{\left(\nabla\hat{f}(\boldsymbol{p}_1) - \nabla\hat{f}(\boldsymbol{p}_2)\right)^T(\boldsymbol{p}_1 - \boldsymbol{p}_2)}{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2} = \frac{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|_H^2}{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2} \in [c_2, c_1]$$

for bounding ψ_i for $i > 0$, and the bound for ψ_0 is directly from the bounds of γ_t. The combined bound is therefore $[\min\{c_2, \delta\}, \max\{c_1, \|X^TX\|L\}]$. Next, we show that the final ψ_i is always upper-bounded. The right-hand side of (11) is equivalent to the following:

$$\arg\min_{\boldsymbol{d}} \hat{Q}_{\psi_i}(\boldsymbol{d}) \coloneqq \nabla\hat{f}(\boldsymbol{p}^{(i)})^T\boldsymbol{d} + \frac{\psi_i\|\boldsymbol{d}\|^2}{2} + \hat{g}(\boldsymbol{d}+\boldsymbol{p}) - \hat{g}(\boldsymbol{p}). \tag{25}$$

Denote the optimal solution by $\boldsymbol{d}^*$, then we have $\boldsymbol{p}^{(i+1)} = \boldsymbol{p}^{(i)} + \boldsymbol{d}^*$. Because H is upper-bounded by c_1, we have that $\nabla\hat{f}$ is c_1-Lipschitz continuous. Therefore, using Lemma 12 of Lee and Wright [9], we get

$$\hat{Q}_{\psi_i}(\boldsymbol{d}^*) \le -\frac{\psi_i}{2}\|\boldsymbol{d}^*\|^2. \tag{26}$$

We then have from c_1-Lipschitz continuity of $\nabla\hat{f}$ that

$$\begin{aligned}&Q\left(\boldsymbol{p}^{(i+1)}\right) - Q\left(\boldsymbol{p}^{(i)}\right)\\ &\le \nabla\hat{f}(\boldsymbol{p}^{(i)})^T\left(\boldsymbol{p}^{(i+1)} - \boldsymbol{p}^{(i)}\right) + \frac{c_1}{2}\left\|\boldsymbol{p}^{(i+1)} - \boldsymbol{p}^{(i)}\right\|^2\\ &\quad + \hat{g}\left(\boldsymbol{p}^{(i+1)}\right) - \hat{g}\left(\boldsymbol{p}^{(i)}\right)\\ &\overset{(25)}{=} \hat{Q}_{\psi_i}(\boldsymbol{d}^*) - \frac{\psi_i}{2}\|\boldsymbol{d}^*\|^2 + \frac{c_1}{2}\|\boldsymbol{d}^*\|^2\\ &\overset{(26)}{\le} \left(\frac{c_1}{2} - \psi_i\right)\|\boldsymbol{d}^*\|^2.\end{aligned}$$

Therefore, whenever

$$\frac{c_1}{2} - \psi_i \le -\frac{\sigma_0\psi_i}{2},$$

(13) holds. This is equivalent to

$$\psi_i \ge \frac{c_1}{2-\sigma_0}.$$

Since $\sigma_0 \in (0,1)$, we must have $c_1/(2-\sigma_0) \in (c_1/2, c_1)$, Note that the initialization of ψ_i is upper-bounded by c_1 for all $i > 1$, so the final ψ_i is upper bounded by $2c_1$. Together with the first iteration that we start with $\psi_0 = \gamma_t$, we have that ψ_i are always upper-bounded by $\max\{2c_1, \gamma_t\}$, and we have already proven γ_t is upper-bounded by $\|X^TX\|L$.

(3) We note that since Q is c_2-strongly convex, the following condition holds.

$$\frac{\min_{\boldsymbol{s}\in\nabla\hat{f}\left(\boldsymbol{p}^{(i+1)}\right)+\partial\hat{g}\left(\boldsymbol{p}^{(i+1)}\right)}\|\boldsymbol{s}\|^2}{2c_2} \ge Q\left(\boldsymbol{p}^{(i+1)}\right) - Q^* \tag{27}$$

On the other hand, from the optimality condition of (25), we have that

$$\psi_i\boldsymbol{d}^* = \nabla\hat{f}\left(\boldsymbol{p}^{(i)}\right) + \boldsymbol{s}_{i+1}, \tag{28}$$

for some

$$\boldsymbol{s}_{i+1} \in \partial\hat{g}\left(\boldsymbol{p}^{(i+1)}\right).$$

Therefore,

$$\begin{aligned}&Q\left(\boldsymbol{p}^{(i+1)}\right) - Q^*\\ &\overset{(27)}{\le} \frac{1}{2c_2}\left\|\nabla\hat{f}\left(\boldsymbol{p}^{(i+1)}\right) - \nabla\hat{f}\left(\boldsymbol{p}^{(i)}\right) + \nabla\hat{f}\left(\boldsymbol{p}^{(i)}\right) + \boldsymbol{s}_{i+1}\right\|^2\\ &\overset{(28)}{\le} \frac{1}{c_2}\left\|\nabla\hat{f}\left(\boldsymbol{p}^{(i+1)}\right) - \nabla\hat{f}\left(\boldsymbol{p}^{(i)}\right)\right\|^2 + \|\psi_i\boldsymbol{d}^*\|^2\\ &\le \frac{1}{c_2}\left(c_1^2 + \psi^2\right)\|\boldsymbol{d}^*\|^2.\end{aligned} \tag{29}$$

By combining (13) and (29), we obtain

$$\begin{aligned}Q\left(\boldsymbol{p}^{(i+1)}\right) - Q\left(\boldsymbol{p}^{(i)}\right) &\le -\frac{\sigma_0\psi_i}{2}\|\boldsymbol{d}^*\|^2\\ &\le -\frac{\sigma_0\psi_i}{2}\frac{c_2}{c_1^2+\psi^2}\left(Q\left(\boldsymbol{p}^{(i+1)}\right) - Q^*\right).\end{aligned}$$

Rearranging the terms, we obtain

$$\left(1 + \frac{c_2\sigma_0\psi_i}{2(c_1^2+\psi^2)}\right)\left(Q\left(\boldsymbol{p}^{(i+1)}\right) - Q^*\right) \le Q\left(\boldsymbol{p}^{(i)}\right) - Q^*,$$

showing Q-linear convergence of SpaRSA, with

$$\eta = \left(1 + \frac{c_2\sigma_0\psi_i}{2\left(c_1^2+\psi_i^2\right)}\right)^{-1} \in [0,1]. \qquad \square$$