Limited-memory Common-directions Method for Distributed L1-regularized Linear Classification

Wei-Lin Chiang* Yu-Sheng Li† Ching-pei Lee‡ Chih-Jen Lin§

Abstract

For distributed linear classification, L1 regularization is useful because of a smaller model size. However, with the non-differentiability, it is more difficult to develop efficient optimization algorithms. In the past decade, OWLQN has emerged as the major method for distributed training of L1 problems. In this work, we point out issues in OWLQN's search directions. Then we extend the recently developed limited-memory common-directions method for L2-regularized problems to L1 scenarios. Through a unified interpretation of batch methods for L1 problems, we explain why OWLQN has been a popular method and why our method is superior in distributed environments. Experiments confirm that the proposed method is faster than OWLQN in most situations.

1 Introduction

Given training data $(y_i, \boldsymbol{x}_i), i = 1, \ldots, l$ with label $y_i = \pm 1$ and feature vector $\boldsymbol{x}_i \in \mathbb{R}^n$, we consider L1-regularized linear classification problem

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \equiv \|\boldsymbol{w}\|_1 + C \sum_{i=1}^{l} \xi(y_i \boldsymbol{w}^T \boldsymbol{x}_i), \tag{1.1}$$

where $\|\boldsymbol{w}\|_1 = \sum_{j=1}^{n} |w_j|$ and ξ is a differentiable and convex loss function. Here we consider the logistic loss

$$\xi(z) = \log(1 + e^{-z}).$$

For large applications, (1.1) is often considered because of the model sparsity. Many optimization techniques have been proposed to solve (1.1); see, for example, the comparison in [13]. Generally (1.1) is more difficult than L2-regularized problems because of the non-differentiability.

For large-scale data, distributed training is needed. Although for some applications an online method like [9] is effective, for many other applications batch methods are used to more accurately solve the optimization problem. Currently, OWLQN [1], an extension of a limited-memory quasi-Newton method (LBFGS) [7], is the most commonly used distributed method for L1-regularized classification. For example, it is the main linear classifier in Spark MLlib [10], a popular machine learning tool on Spark. OWLQN's popularity comes from several attributes. First, in a single-machine setting, the comparison [13] shows that OWLQN is competitive among solvers for L1-regularized logistic regression. Second, while coordinate descent (CD) or its variants [14] are state-of-the-art, they are inherently sequential and are difficult to parallelize in distributed environments. Even if they have been modified for distributed training (e.g., [8]), usually a requirement is that data points are stored in a feature-wise manner. In contrast, as we will see in the discussion in this paper, OWLQN is easier to parallelize and allows data to be stored either in an instance-wise or a feature-wise way.

The motivation of this work is to study why OWLQN has been successful and whether we can develop a better distributed training method. We begin with showing in Section 2 how OWLQN is extended from the method LBFGS [7]. From that we point out some issues of OWLQN's direction finding at each iteration. Then in Section 3 we extend a recently developed limited-memory common-directions method [12, 6] for L2-regularized problems to the L1 setting. In Section 4, through a unified interpretation of methods for L1 problems, we explain why our proposed method is more principled than OWLQN. Although from Section 3 our method is slightly more expensive per iteration, the explanation in Section 4 and the past results on L2-regularized problems indicate the potential of fewer iterations. For distributed implementations, in Section 5 we show that OWLQN and our method have similar communication costs per iteration. Therefore, our method can be very useful for distributed training because the communication occupies a significant portion of the total running time after parallelizing the computation, and our method needs fewer total iterations. The result is confirmed through detailed experiments in Section 6.

*National Taiwan University. b02902056@ntu.edu.tw. Part of this work was done when the first, the second, and the last authors visited Alibaba, Inc. This work is also partially supported by MOST of Taiwan grant 104-2221-E-002-047-MY3.

†National Taiwan University. b03902086@ntu.edu.tw

‡University of Wisconsin-Madison. ching-pei@cs.wisc.edu

§National Taiwan University. cjlin@csie.ntu.edu.tw

The proposed method has been implemented in MPI-LIBLINEAR (`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/`). Supplementary materials and programs used in this paper are available at `http://www.csie.ntu.edu.tw/~cjlin/papers/l-commdir-l1/`.

2 OWLQN: Orthant-Wise Limited-memory Quasi-Newton Method

We introduce OWLQN and discuss its issues.

2.1 Limited-memory BFGS (LBFGS) Method.

OWLQN is an extension of the method LBFGS [7] for the following L2-regularized problem.

$$f(\boldsymbol{w}) \equiv \frac{\boldsymbol{w}^T\boldsymbol{w}}{2} + L(\boldsymbol{w}), \tag{2.2}$$

where

$$L(\boldsymbol{w}) \equiv C\sum_{i=1}^{l} \xi(y_i \boldsymbol{w}^T \boldsymbol{x}_i). \tag{2.3}$$

To minimize $f(\boldsymbol{w})$, Newton methods are commonly used. At the current $\boldsymbol{w}_k$, a direction is obtained by

$$\boldsymbol{d} = -\nabla^2 f(\boldsymbol{w}_k)^{-1}\nabla f(\boldsymbol{w}_k). \tag{2.4}$$

Because calculating $\nabla^2 f(\boldsymbol{w}_k)$ and its inverse may be expensive, quasi-Newton techniques have been proposed to obtain an approximate direction

$$\boldsymbol{d} = -B_k \nabla f(\boldsymbol{w}_k), \text{ where } B_k \approx \nabla^2 f(\boldsymbol{w}_k)^{-1}.$$

BFGS [11] is a representative quasi-Newton technique that uses information from all past iterations and the following update formula

$$B_k = V_{k-1}^T B_{k-1} V_{k-1} + \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T,$$

where

$$V_{k-1} \equiv I - \rho_{k-1}\boldsymbol{u}_{k-1}\boldsymbol{s}_{k-1}^T, \quad \rho_{k-1} \equiv 1/(\boldsymbol{u}_{k-1}^T\boldsymbol{s}_{k-1}),$$
$$\boldsymbol{s}_{k-1} \equiv \boldsymbol{w}_k - \boldsymbol{w}_{k-1}, \quad \boldsymbol{u}_{k-1} \equiv \nabla f(\boldsymbol{w}_k) - \nabla f(\boldsymbol{w}_{k-1}),$$

and I is the identity matrix.

To reduce the cost, LBFGS [7] proposes using information from the previous m iterations. From the derivation in [7], the direction $\boldsymbol{d}$ can be efficiently obtained by $2m$ inner products using columns in the following matrix

$$P = \left[\boldsymbol{s}_{k-m}, \boldsymbol{u}_{k-m}, \ldots, \boldsymbol{s}_{k-1}, \boldsymbol{u}_{k-1}\right] \in \mathbb{R}^{n\times 2m}. \tag{2.5}$$

After obtaining $\boldsymbol{d}$, a line search ensures the sufficient decrease of the function value (details not shown). Algorithm I in the supplementary materials summarizes the procedure of LBFGS.

2.2 Modification from LBFGS to OWLQN.

OWLQN extends LBFGS by noticing that instead of the optimality condition

$$\nabla f(\boldsymbol{w}) = \boldsymbol{0}$$

for smooth optimization, for L1 problems in (1.1), $\boldsymbol{w}$ is a global optimum if and only if the projected gradient (PG) is zero.

$$\nabla^{\mathrm{P}} f(\boldsymbol{w}) = \boldsymbol{0},$$

where for $j = 1, \ldots, n$,

$$\nabla_j^{\mathrm{P}} f(\boldsymbol{w}) \equiv \tag{2.6}$$
$$\begin{cases} \nabla_j L(\boldsymbol{w}) + 1 & \text{if } w_j > 0, \text{ or } w_j = 0, \nabla_j L(\boldsymbol{w}) + 1 < 0, \\ \nabla_j L(\boldsymbol{w}) - 1 & \text{if } w_j < 0, \text{ or } w_j = 0, \nabla_j L(\boldsymbol{w}) - 1 > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The concept of projected gradient is from bound-constrained optimization. If we let

$$\boldsymbol{w} = \boldsymbol{w}^+ - \boldsymbol{w}^-,$$

an equivalent bound-constrained problem of (1.1) is

$$\min_{\boldsymbol{w}^+, \boldsymbol{w}^-} \sum_j w_j^+ + \sum_j w_j^- + C\sum_{i=1}^{l} \xi(y_i(\boldsymbol{w}^+ - \boldsymbol{w}^-)^T\boldsymbol{x}_i)$$
$$\text{subject to } w_j^+ \geq 0,\ w_j^- \geq 0,\ \forall j.$$

Roughly speaking, the projected gradient indicates whether we can update w_j by a gradient descent step or not. For example, if

$$(\boldsymbol{w}_k)_j = 0 \text{ and } \nabla_j L(\boldsymbol{w}_k) + 1 < 0, \tag{2.7}$$

then

$$(\boldsymbol{w}_k)_j - \alpha(\nabla_j L(\boldsymbol{w}_k) + 1) > 0, \forall \alpha > 0$$

remain in the orthant of $\boldsymbol{w}_j \geq 0$ (or $\boldsymbol{w}_j^+ \geq 0, \boldsymbol{w}_j^- = 0$). On this face f is differentiable with respect to w_j and

$$\nabla_j f(\boldsymbol{w}_k) = \nabla_j L(\boldsymbol{w}_k) + 1 \tag{2.8}$$

exists. Therefore, if we update w_j along the direction of $-\nabla_j f(\boldsymbol{w}_k)$ with a small enough step size, the objective value will decrease. Hence if (2.7) holds, the projected gradient is defined to be the value in (2.8).

From the above explanation, the projected gradient roughly splits all variables to two sets: an active one containing elements that might still be modified, and a non-active one including elements that should remain the same. By defining the following active set

$$A \equiv \{j \mid \nabla_j^{\mathrm{P}} f(\boldsymbol{w}_k) \neq 0\}, \tag{2.9}$$

OWLQN simulates LBFGS on the face characterized by the set A, and proposes the following modifications.

1. Because $\nabla f(\boldsymbol{w}_k)$ does not exist, the direction $\boldsymbol{d}$ is obtained by using the projected gradient

$$\boldsymbol{d} = -B_k \nabla^{\mathrm{P}} f(\boldsymbol{w}_k). \tag{2.10}$$

They apply the same procedure of $\mathcal{O}(m)$ vector operations (Algorithm II in the supplementary materials) to get $\boldsymbol{d}$, but $\boldsymbol{u}_{k-1}$ is replaced by

$$\boldsymbol{u}_{k-1} \equiv \nabla L(\boldsymbol{w}_k) - \nabla L(\boldsymbol{w}_{k-1}).$$

Namely, now B_k is an approximation of $\nabla^2 L(\boldsymbol{w}_k)^{-1}$ but not $\nabla^2 f(\boldsymbol{w}_k)^{-1}$.
2. The search direction is aligned with $-\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$:

$$d_j \leftarrow 0 \text{ if } -d_j \nabla_j^{\mathrm{P}} f(\boldsymbol{w}_k) \le 0. \tag{2.11}$$

3. In line search, they ensure that the new value stays at the same orthant as the original one.

$$w_j \leftarrow \begin{cases} \max(0, w_j + \alpha d_j) & \text{if } w_j > 0 \text{ or} \\ & w_j = 0, \nabla_j L(\boldsymbol{w}) + 1 < 0 \\ \min(0, w_j + \alpha d_j) & \text{if } w_j < 0 \text{ or} \\ & w_j = 0, \nabla_j L(\boldsymbol{w}) - 1 > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

2.3 Complexity. In one iteration, the following operations are conducted.

1. From (2.13), the evaluation of $\nabla L(\boldsymbol{w}_k)$ takes $\mathcal{O}(\#\text{nnz})$, where #nnz is the number of non-zeros in the training data.

$$\nabla L(\boldsymbol{w}_k) = C \sum\nolimits_{i=1}^{l} (\sigma(y_i \boldsymbol{w}_k^T \boldsymbol{x}_i) - 1) y_i \boldsymbol{x}_i. \tag{2.13}$$

2. At each step of line search, calculating $\boldsymbol{w}_k^T \boldsymbol{x}_i, \forall i$ needs $\mathcal{O}(\#\text{nnz})$.
3. For obtaining the direction in (2.10), the $2m$ inner products take $\mathcal{O}(mn)$.

Therefore, the cost per OWLQN iteration is

$$(1 + \#\text{line-search steps}) \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(mn). \tag{2.14}$$

For L2-regularized problems, a trick can be applied to significantly reduce the line-search cost. However, this trick is not applicable for L1-regularized problems because of the max and min operations in (2.12); see more details in Section IV of the supplementary materials. Therefore, ensuring a small number of line-search steps is very essential.

2.4 Issues of OWLQN. Although efficient in practice, OWLQN still possesses several issues in different aspects, as we describe below. First, it is known that this method lacks convergence guarantee, though a slightly modified algorithm with asymptotic convergence is proposed recently [5]. Second, under an active set A, we would like to get a good direction by minimizing the following second-order approximation.

$$\min_{\boldsymbol{d}_A} \quad \frac{1}{2} \boldsymbol{d}_A^T \nabla_{AA}^2 L(\boldsymbol{w}_k) \boldsymbol{d}_A + \nabla_A^{\mathrm{P}} f(\boldsymbol{w}_k)^T \boldsymbol{d}_A, \tag{2.15}$$

where $L(\boldsymbol{w}_k)$ is defined in (2.3). Thus a quasi-Newton setting should approximate

$$\nabla_{AA}^2 L(\boldsymbol{w}_k)^{-1} \text{ rather than } (\nabla^2 L(\boldsymbol{w}_k)^{-1})_{AA},$$

but the latter is closer to what OWLQN uses. We see that the mapping to A by an alignment with $-\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$ in (2.11) is conducted after the direction finding in (2.10). Indeed we observe that using a larger m in OWLQN does not help improve the convergence in terms of number of iterations, while it does for LBFGS on smooth problems, indicating that direction finding in OWLQN is not ideal.

3 Our Limited-memory Common-directions Algorithm for L1-regularized Classification

We extend the limited-memory common-directions method for L2-regularized problems [12, 6] to L1-regularized classification.

3.1 Past Developments for L2-regularized Problems. To solve (2.2), the method [12] considers an orthonormal basis P_k of all past gradients (called *common directions*) and obtains the direction $\boldsymbol{d} = P_k \boldsymbol{t}$ by solving the following second-order sub-problem.

$$\min_{\boldsymbol{t}} \nabla f(\boldsymbol{w}_k)^T P_k \boldsymbol{t} + \frac{1}{2} \boldsymbol{t}^T P_k^T \nabla^2 f(\boldsymbol{w}_k) P_k \boldsymbol{t}, \tag{3.16}$$

or equivalently, the following linear system:

$$P_k^T \nabla^2 f(\boldsymbol{w}_k) P_k \boldsymbol{t} = -P_k^T \nabla f(\boldsymbol{w}_k). \tag{3.17}$$

Therefore, in contrast to (2.4) for obtaining a Newton direction, here the direction is restricted to be a linear combination of P_k's columns. For (2.2), we can derive

$$\nabla^2 f(\boldsymbol{w}_k) = I + C X^T D_{\boldsymbol{w}_k} X, \tag{3.18}$$

where $D_{\boldsymbol{w}_k}$ with $(D_{\boldsymbol{w}_k})_{ii} = \xi''(y_i \boldsymbol{w}_k^T \boldsymbol{x}_i)$ is a diagonal matrix and $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T \in \mathbb{R}^{l \times n}$ includes all training instances. Then we get

$$P_k^T \nabla^2 f(\boldsymbol{w}_k) P_k = P_k^T P_k + C(XP_k)^T D_{\boldsymbol{w}_k} (XP_k). \tag{3.19}$$

Note that $P_k^T P_k = I$ from the orthonormality of P_k. In [12], they devise a technique to cheaply maintain XP_k.

Then (3.17) is a linear system of k variables and can be easily solved if the number of iterations is not large.

To further save the cost, following the idea of LBFGS in Section 2.1, [6] proposes using only information from the past m iterations. They checked several choices of common directions, but for simplicity we consider the most effective one of using past m gradients, past m steps, and the current gradient.

$$P_k = [\nabla f(\boldsymbol{w}_{k-m}), \boldsymbol{s}_{k-m}, \ldots, \nabla f(\boldsymbol{w}_{k-1}), \boldsymbol{s}_{k-1}, \nabla f(\boldsymbol{w}_k)] \in R^{n\times(2m+1)}. \tag{3.20}$$

Because P_k no longer contains orthonormal columns, $P_k^T P_k \neq I$. Similar to [2], [6] developed an $\mathcal{O}(mn)$ technique to easily update $P_{k-1}^T P_{k-1}$ to $P_k^T P_k$. The cost is about the same as the $2m$ vector operations in LBFGS for obtaining the update direction. (See Algorithm II in the supplementary materials.) After solving (3.16) and finding a direction $P_k\boldsymbol{t}$, they conduct a line search procedure to find a suitable step size. Experiments in [6] show their method is superior to LBFGS.

To maintain XP_k, we need two new vectors

$$X\boldsymbol{s}_{k-1} \text{ and } X\nabla f(\boldsymbol{w}_k),$$

each of which expensively costs $\mathcal{O}(\#\text{nnz})$. If we maintain $X\boldsymbol{w}_{k-1}$ from the function evaluation of the previous iteration, then

$$X\boldsymbol{s}_{k-1} = X\boldsymbol{w}_k - X\boldsymbol{w}_{k-1}$$

can be done more cheaply in $\mathcal{O}(l)$. Thus $X\nabla f(\boldsymbol{w}_k)$ is the only expensive operation for maintaining XP_k. Therefore, in comparison with LBFGS, at each iteration extra costs include

- $\mathcal{O}(\#\text{nnz})$ for $X\nabla f(\boldsymbol{w}_k)$ in the matrix XP_k,
- $\mathcal{O}(lm^2)$ for $(XP_k)^T D_{\boldsymbol{w}_k}(XP_k)$,
- $\mathcal{O}(mn)$ for the right-hand side $-P_k^T\nabla f(\boldsymbol{w}_k)$ of the linear system (3.17),
- $\mathcal{O}(m^3)$ for solving (3.17), and
- $\mathcal{O}(mn)$ for computing $\boldsymbol{d} = P_k\boldsymbol{t}$ after obtaining $\boldsymbol{t}$.

3.2 Modification for L1-regularized Problems. We propose the following simple modifications.

1. In the sub-problem (3.16), $\nabla f(\boldsymbol{w}_k)$ or $\nabla^2 f(\boldsymbol{w}_k)$ may not exist, so instead we use $\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$ and $\nabla^2 L(\boldsymbol{w}_k)$, respectively. Thus (3.16) becomes

$$\min_{\boldsymbol{t}} \nabla^{\mathrm{P}} f(\boldsymbol{w}_k)^T P_k\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T P_k^T \nabla^2 L(\boldsymbol{w}_k) P_k\boldsymbol{t}. \tag{3.21}$$

2. In the matrix P_k of common directions, we use past projected gradients

$$\nabla^{\mathrm{P}} f(\boldsymbol{w}_{k-m}), \ldots, \nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$$

rather than gradients.

3. Similar to the setting in OWLQN, the search direction is aligned with $\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$; see (2.11). Furthermore, during line search, the new point stays at the same orthant as the original one; see (2.12).

In Section 4 we will argue that the resulting Algorithm III (in supplementary materials) is more reasonable than OWLQN.

3.3 Complexity. In Sections 3.1-3.2, we have discussed operations related to using the common directions. Besides them, function/gradient evaluations are needed. Therefore, the cost per iteration is

$$(2 + \#\text{line-search steps}) \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(lm^2) + \mathcal{O}(mn) + \mathcal{O}(m^3). \tag{3.22}$$

With $m \ll l$ and $m \ll n$, among the last three terms, $\mathcal{O}(lm^2)$ is the dominant one. In some situations, it may even be the most expensive term in (3.22). For example, if line search terminates in one or two steps and the data set is highly sparse such that

$$\#\text{nnz} \approx lm, \text{ then } lm^2 \approx m \times \#\text{nnz}$$

becomes the bottleneck. Fortunately, the $\mathcal{O}(lm^2)$ term comes from dense matrix-matrix products for constructing the linear system in (3.17). They can be efficiently conducted by using optimized BLAS. Therefore, in general the first term in (3.22) for function/gradient evaluations and $X\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$ is still the main bottleneck.

A comparison between (3.22) and OWLQN's complexity in (2.14) shows that at every iteration, we need one more $\mathcal{O}(\#\text{nnz})$ operation for calculating $X\nabla^{\mathrm{P}} f(\boldsymbol{w}_k)$. Therefore, if line search takes only one step per iteration (i.e. #line-search steps = 1; see more details in Section 3.4), then we have the following difference between OWLQN and our proposed method:

$$2 \times \mathcal{O}(\#\text{nnz}) \text{ versus } 3 \times \mathcal{O}(\#\text{nnz}). \tag{3.23}$$

This seems to indicate that each iteration of our method is 50% more expensive than that of OWLQN. However, the three $\mathcal{O}(\#\text{nnz})$ operations include

$$\text{one } X^T \times (\text{a vector}) \text{ and two } X \times (\text{a vector}).$$

In Section 5 we show that in a distributed environment, the communication cost of $X^T \times$ (a vector) is much higher. Thus ours and OWLQN have similar communication cost per iteration. Then the higher computational cost may pay off if the number of iterations becomes smaller.

3.4 Reducing the Number of Line-Search Steps. We have shown in Section 3.3 that each line-search step needs $\mathcal{O}(\#\text{nnz})$ cost for calculating the function value. Because a search starting from $\alpha = 1$ may

Algorithm 1 An optimization framework for L1-regularized problems.

1: **while** true **do**
2: Compute projected gradient $\nabla^{\mathrm{P}} f(\boldsymbol{w})$ by (2.6)
3: Find the set A in (2.9)
4: Get a direction $\boldsymbol{d}$ with $d_j = 0, \forall j \notin A$
5: Align the direction $\boldsymbol{d}$ with $-\nabla^{\mathrm{P}} f(\boldsymbol{w})$; see (2.11)
6: Line search (and update model parameters)

result in many steps, we propose using α of the previous iteration as the initial step size. However, to avoid slow convergence of taking too small step sizes, once in several iterations we double the α of the previous iteration as the initial step size.

4 A Unified Interpretation of Methods for L1-regularized Classification

We give a unified interpretation of methods for L1-regularized classification. It helps to explain first why our proposed method is superior to OWLQN, and second, why OWLQN has been popularly used so far.

In Section 2.2 we explain that the L1-regularized problem is equivalent to a bound-constrained problem. Most existing methods for bound-constrained optimization conduct the following two steps at each iteration.

- Identify a face to be used for getting a direction.
- Under a given face, obtain a search direction by unconstrained optimization techniques.

The idea is that if we are at the same (or a similar) face of an optimal point, unconstrained optimization techniques can be effectively applied there. In general the identified face is related to non-zero elements of the projected gradient $\nabla^{\mathrm{P}} f(\boldsymbol{w})$. For an easy discussion, here we do not touch past developments on face selection; instead, following OWLQN, we consider the set A in (2.9) that consists of all non-zero elements in $\nabla^{\mathrm{P}} f(\boldsymbol{w})$. We then focus on obtaining a suitable direction on A. A framework under this setting is in Algorithm 1. Two commonly used search directions are

1. Negative projected gradient direction $-\nabla^{\mathrm{P}}_A f(\boldsymbol{w})$.
2. Newton direction by minimizing (2.15)

$$-\nabla^2_{AA} L(\boldsymbol{w})^{-1} \nabla^{\mathrm{P}}_A f(\boldsymbol{w}).$$

Note that a convex loss implies only that $\nabla^2 L(\boldsymbol{w})$ is positive semi-definite. For simplicity, we assume it is positive definite and therefore invertible.

For the Newton direction, because $\nabla^2_{AA} L(\boldsymbol{w})$ may be too large to be stored, in linear classification, commonly the special structure in (3.18) (without the identity term) is considered so that a conjugate gradient (CG) procedure is used to solve the linear system

$$\nabla^2_{AA} L(\boldsymbol{w}) \boldsymbol{d}_A = -\nabla^{\mathrm{P}}_A f(\boldsymbol{w}). \tag{4.24}$$

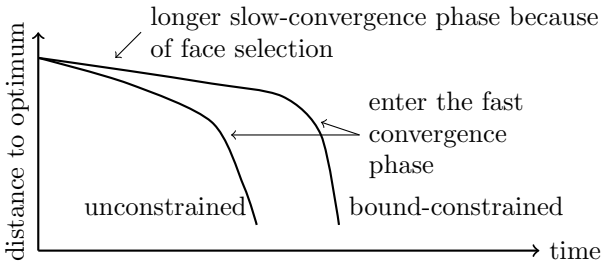


Figure 1: An illustration of Newton methods for unconstrained and bound-constrained optimization.

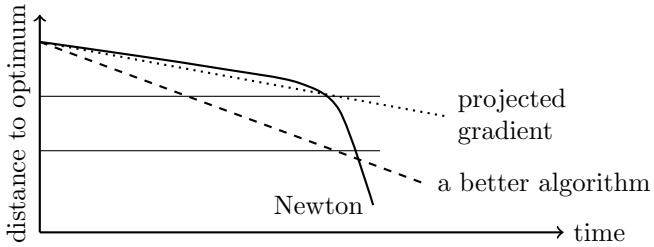


Figure 2: An illustration of optimization methods for L1 problems. The two horizontal lines indicate suitable range to stop for machine learning applications.

CG involves a sequence of matrix-vector products and each is performed by

$$(X_{:,A})^T (D_{\boldsymbol{w}} (X_{:,A} \boldsymbol{v})), \tag{4.25}$$

where $D_{\boldsymbol{w}}$ with $(D_{\boldsymbol{w}})_{ii} = \xi''(y_i \boldsymbol{w}^T \boldsymbol{x}_i)$ is a diagonal matrix and $\boldsymbol{v}$ is some vector involved in CG procedure.

We compare the costs of using the above two directions by assuming that line search terminates in one step. Their complexities per iteration are respectively

$$2 \times \mathcal{O}(\#\mathrm{nnz}) \quad \text{and} \quad (2 + \#\mathrm{CG}) \times \mathcal{O}(\#\mathrm{nnz}).$$

The above cost comes from one function and one gradient evaluation, and for the Newton method, each matrix-vector product in (4.25) requires $\mathcal{O}(\#\mathrm{nnz})$. One may argue that (4.25) costs $\mathcal{O}(\#\mathrm{nnz} \times |A|/l)$ because only $|A|$ columns of X are used. We consider $\mathcal{O}(\#\mathrm{nnz})$ because, firstly, the analysis is simpler, and secondly, if X is stored in an instance-wise manner, $\mathcal{O}(\#\mathrm{nnz})$ is needed. Clearly the cost of using a Newton direction is much higher. Unfortunately, the higher cost may not pay off because of the following reasons.

1. The second-order approximation is accurate only near a solution, so a Newton direction in the early stage may not be better than the negative gradient direction.
2. The face A is still very different from that of the convergent point, so a good direction obtained on an unsuitable face is not useful.

The first issue has occurred in unconstrained optimization: a Newton method has fast final convergence, but the early convergence is slow. The second issue, spe-

cific to bound-constrained optimization, makes the decision of using an expensive direction or not even more difficult. We give an illustration of Newton methods for unconstrained and bound-constrained optimization in Figure 1. For the method of using projected gradient, it not only inherits the slow convergence of gradient descent methods for unconstrained optimization, but also has the issue of face identification. Therefore, the two methods behave like the curves shown in Figure 2. Neither is satisfactory in practice. What we hope is something in between: the direction should be better than negative projected gradient, but not as expensive as the Newton direction. The dashed curve in Figure 2 is such an example. It is slower than Newton at the final stage, but for machine learning applications, often the optimization procedure can stop earlier, for example, at the region between the two horizontal lines in Figure 2. Then the new method is the fastest. To have a method in between, we can either

1. improve the projected gradient direction, or
2. reduce the cost of obtaining a Newton direction.

OWLQN is an example. Its quasi-Newton direction needs $\mathcal{O}(mn)$ additional cost, but should be better than the projected gradient. Results of OWLQN with similar trends shown in Figure 2 have been confirmed empirically in [13]. Thus our analysis illustrates why OWLQN has been a popular method for L1 problems.

From the above discussion, many algorithms can be proposed. For example, to reduce the cost of obtaining the Newton direction, we can take fewer CG steps to loosely solve the linear system (4.24). For example, if $\#\text{CG} = m$, then the cost per iteration is reduced to

$$(2+m) \times \mathcal{O}(\#\text{nnz}).$$

Next we derive the proposed limited-memory common-directions method from the viewpoint of reducing the cost of obtaining a Newton direction. By restricting the direction to be a linear combination of some vectors (i.e., columns in P), (2.15) becomes

$$\min_{\boldsymbol{t}} \nabla^{\text{P}}_A f(\boldsymbol{w}_k)^T (P\boldsymbol{t})_A + \frac{1}{2}((P\boldsymbol{t})_A)^T \nabla^2_{AA} L(\boldsymbol{w}_k)(P\boldsymbol{t})_A.$$

From (3.19), the quadratic term is

$$\begin{aligned} &((P\boldsymbol{t})_A)^T \nabla^2_{AA} L(\boldsymbol{w}_k)(P\boldsymbol{t})_A \\ &= C\boldsymbol{t}^T (X_{:,A}P_{A,:})^T D_{\boldsymbol{w}_k} (X_{:,A}P_{A,:})\boldsymbol{t} \end{aligned} \tag{4.26}$$

If P has m columns, obtaining $X_{:,A}P_{A,:}$, requires m matrix-vector products, each of which costs $\mathcal{O}(\#\text{nnz})$. We then solve a linear system of m variables. The setting is similar to that in Section 3. Therefore, the cost per iteration is

$$(2+m) \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(nm) + \mathcal{O}(lm^2) + \mathcal{O}(m^3).$$

Table 1: A summary of complexity per iteration for all methods, sorted from cheapest to most expensive. We assume #line-search steps = 1. Only leading terms are presented by assuming $m \ll l$ and $m \ll n$.

Method	Cost per iteration
Projected gradient	$2 \times \mathcal{O}(\#\text{nnz})$
OWLQN	$2 \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(mn)$
L-Comm	$3 \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(lm^2)$
L-Comm-Face	$(2+m) \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(lm^2)$
Newton + m CG	$(2+m) \times \mathcal{O}(\#\text{nnz})$

We refer to this approach as "L-Comm-Face." Unfortunately, the cost may be too high. It is as expensive as the Newton method of running m CG steps. We therefore propose an approximation of $X_{:,A}P_{A,:}$ that eventually leads to the proposed method in Section 3. If the past m projected gradients have similar active sets to the current $\nabla^{\text{P}} f(\boldsymbol{w}_k)$, then all m past steps have similar non-zero entries. Therefore,

$$P_{j,:} \approx [0, \ldots, 0] \quad \forall j \notin A$$

implies that

$$X_{:,A}P_{A,:} \approx XP \text{ and } (4.26) \approx \boldsymbol{t}^T P^T \nabla^2 L(\boldsymbol{w}_k) P\boldsymbol{t}. \tag{4.27}$$

Thus the method becomes that in Section 3 by solving (3.21). The complexity is in (3.22), or if #line-search steps = 1, the complexity becomes

$$3 \times \mathcal{O}(\#\text{nnz}) + \mathcal{O}(lm^2) + \mathcal{O}(mn) + \mathcal{O}(m^3).$$

We refer to the method in Section 3 as "L-Comm," which is an approximation of "L-Comm-Face" by (4.27). The cost is now close to that of OWLQN. In Section 2, we showed some issues of OWLQN in approximating the Newton direction. In contrast, our method is more principled. In Table 1, we summarize the complexity per iteration of each discussed method.

5 Distributed Implementation

We show that the proposed method is very useful in distributed environments. To discuss the communication cost, we make the following assumptions.

1. The data set X is split across K nodes in an instance-wise manner: $J_r, \quad r = 1, \ldots, K$ form a partition of $\{1, \ldots, l\}$, and the r-th node stores $(y_i, \boldsymbol{x}_i)$ for $i \in J_r$. In addition, XP is split and stored in the same way as X.
2. The matrix $P \in R^{n \times 2m}$ in (2.5) for OWLQN and $P \in R^{n \times (2m+1)}$ in (3.20) for the common-directions method are stored in the same K nodes in a row-wise manner. Thus $\{1, \ldots, n\}$ are partitioned to $\bar{J}_1, \ldots, \bar{J}_K$.
3. The model vector $\boldsymbol{w}$ and the projected gradient $\nabla^{\text{P}} f(\boldsymbol{w})$ are made available to all K nodes.

Communication cost occurs in the following places.

1. Compute

$$\nabla L(\boldsymbol{w}) = C \bigoplus_{r=1}^{K} (X_{J_r,:})^T \left[\begin{array}{c} \vdots \\ \xi'(y_i \boldsymbol{w}^T \boldsymbol{x}_i) \\ \vdots \end{array} \right]_{i \in J_r},$$

where $\bigoplus$ is the *allreduce* operation that sums up values from nodes and broadcasts the result back. The communication cost is $\mathcal{O}(n)$.[1]

2. After obtaining $\boldsymbol{d}_{\bar{J}_r}$ at node r, we need an *allgather* operation to make the whole direction $\boldsymbol{d}$ available at every node. The communication cost is $\mathcal{O}(n/K)$.
3. At each line-search step, to get the new function value, we need an *allreduce* operation.

$$f(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 + C \bigoplus_{r=1}^{K} \sum_{i \in J_r} \xi(y_i \boldsymbol{w}^T \boldsymbol{x}_i).$$

At each node, the sum of local losses is the only value to be communicated, so the cost is $\mathcal{O}(1)$.

4. (OWLQN only) Because each node now possesses only a sub-matrix of P, in the procedure of obtaining the search direction $\boldsymbol{d}$, $2m+1$ distributed inner products are needed. Details are left to the supplementary materials. For each distributed inner product, the communication cost is $\mathcal{O}(1)$.
5. (L-Comm only) To construct the linear system in (3.17), we need the following *allreduce* operations. The communication cost is $\mathcal{O}(m^2)$.

$$(XP)^T D_{\boldsymbol{w}} (XP) = \bigoplus_{r=1}^{K} (X_{J_r,:}P)^T (D_{\boldsymbol{w}})_{J_r,J_r} (X_{J_r,:}P)$$
$$P^T \nabla^{\mathrm{P}} f(\boldsymbol{w}) = \bigoplus_{r=1}^{K} (P_{\bar{J}_r,:})^T \nabla^{\mathrm{P}}_{\bar{J}_r} f(\boldsymbol{w}).$$

A rough estimate of the communication cost is

$$\text{OWLQN: } \mathcal{O}(n) + \mathcal{O}(n/K) + \mathcal{O}(1) + (2m+1) \times \mathcal{O}(1),$$
$$\text{L-Comm: } \mathcal{O}(n) + \mathcal{O}(n/K) + \mathcal{O}(1) + \mathcal{O}(m^2).$$

For large and sparse data, $n \gg m^2$. The dominant term is $\mathcal{O}(n)$, so the two methods have similar communication costs. This situation is different from that of computational complexity. In (3.23), we state that our method may be 50% more expensive than OWLQN for computing $X\nabla^{\mathrm{P}} f(\boldsymbol{w})$. However, this operation does not involve any communication because a local node simply calculates $X_{J_r,:}\nabla^{\mathrm{P}} f(\boldsymbol{w})$ to update the local $X_{J_r,:}P$. Thus our method is useful in a distributed setting. Its higher cost per iteration becomes a less serious concern because the computation is parallelized and the communication may take a significant portion of the total time. Algorithm VI in the supplementary materials gives details of a distributed version of L-Comm.

[1] The communication cost depends on the implementation of the *allreduce* operation. Here for easy analysis we use the length of a vector at a local node to be transferred to others as the communication cost.

Table 2: Data statistics.

Data set	#instances	#features	#nonzeros
rcv1_test	677,399	47,226	49,556,258
news20	19,996	1,355,191	9,097,916
yahoojp	176,203	832,026	23,506,415
url	2,396,130	3,231,961	277,058,644
yahookr	460,554	3,052,939	156,436,656
KDD2010-b	19,264,097	29,890,096	566,345,888
criteo	45,840,617	1,000,000	1,787,773,969
kdd2012	149,639,105	54,686,452	1,646,030,155

6 Experiments

We consider binary classification data sets listed in Table 2.[2] We implement all methods under the framework of LIBLINEAR [3] (for Section 6.1) or Distributed LIBLINEAR (for Section 6.2). OpenBLAS[3] is used for dense vector/matrix operations.

We check the convergence of optimization methods by showing the number of iterations or running time versus the relative difference to the optimal function value

$$(f(\boldsymbol{w}_k) - f^*)/f^*,$$

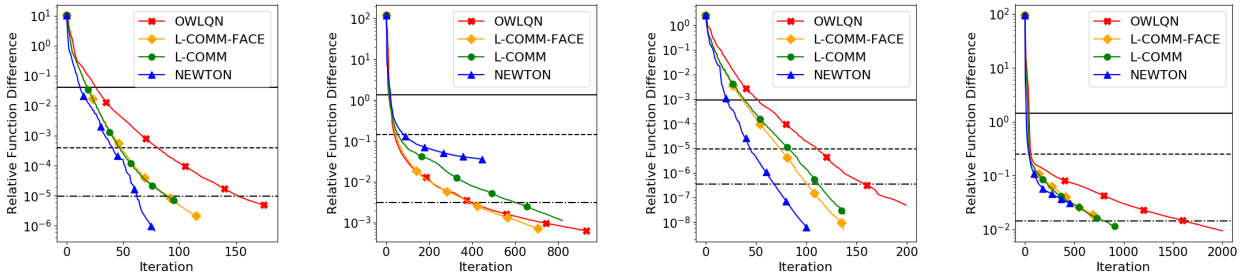
where f^* is an approximate optimal objective value by accurately solving the optimization problem. For the regularization parameter C, we select C_{Best} that achieves the best cross validation accuracy. More details and results of using other C values are in supplementary materials.

6.1 Comparisons on Number of Iterations. From Section 4, both OWLQN and L-Comm are designed to approximate the Newton direction because conceptually better directions should lead to a smaller number of iterations. Without considering the cost per iteration, we begin with checking this result by comparing the following methods discussed in Section 4.

- OWLQN [1]: We consider $m = 10$.
- L-COMM-FACE: We use information from the past 10 iterations to have common directions.
- L-COMM: The proposed method in Section 3, which according to the discussion in Section 4 is an approximation of L-COMM-FACE.
- NEWTON: We run 50 CG steps at each iteration. This setting simulates the situation where the full Newton direction on the face is obtained.

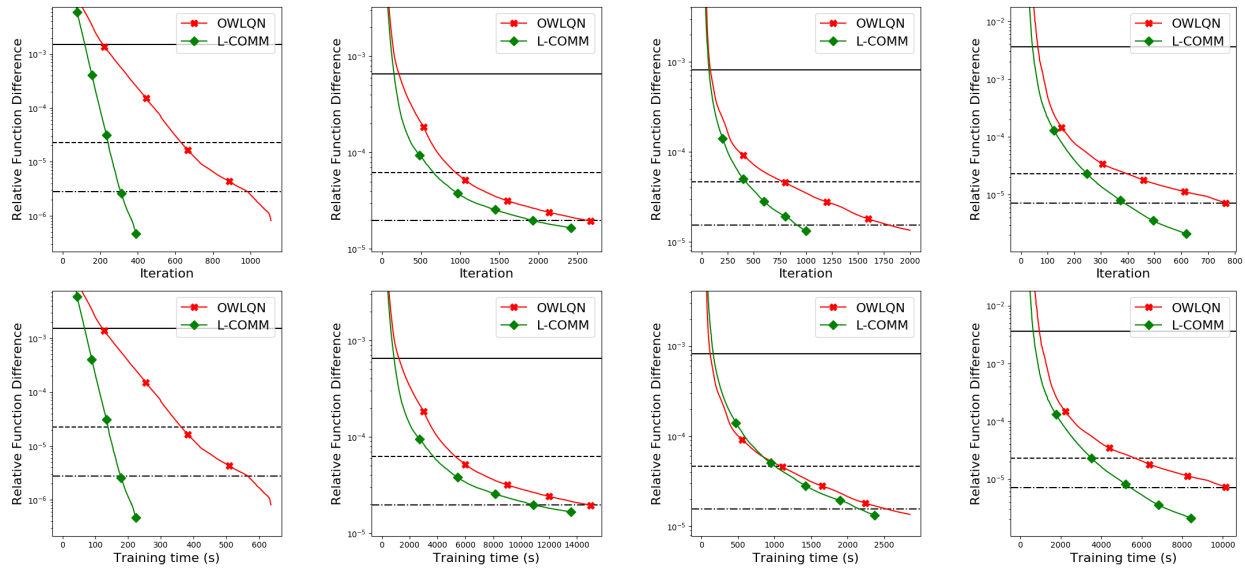
[2] All data sets except `yahoojp` and `yahookr` are from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

[3] `http://www.openblas.net/`



(a) rcv1_test (b) news20 (c) yahoojp (d) url

Figure 3: Iteration versus the difference to the optimal value. The horizontal lines indicate when OWLQN meets the stopping condition $\|\nabla^{\mathrm{P}} f(\boldsymbol{w})\| \leq \epsilon \frac{\min(\#\text{positive},\#\text{negative})}{l} \|\nabla^{\mathrm{P}} f(\mathbf{0})\|$, with $\epsilon = 10^{-2}$, 10^{-3}, and 10^{-4}.



(a) yahookr (b) KDD2010-b (c) criteo (d) kdd2012

Figure 4: Comparison of different algorithms by using 32 nodes. Upper: iterations. Lower: running time in seconds. Other settings are the same as Figure 3.

All these methods find a direction on the face decided by the project gradient and conduct line search to find a suitable step size.[4] Comparison results are in Figure 3 and in supplementary materials. We make the following observations.

- NEWTON is generally among the best, but not always (e.g., news20). A possible explanation is that these methods are not compared under a fixed face. Instead, each method goes through a different sequence of faces. Thus a method of using a better direction locally may not be the best globally. Nevertheless, the overall good performance of NEWTON indicates that one should consider a better direction under any given face if possible. Further, the shape of NEWTON's curve is close to the illustration in Figure 1 for smaller C (see supplementary materials), but for larger C, it is not that close. An explanation is that for larger C, the problem is more difficult and NEWTON is still under the stage of finding a suitable face.
- The convergence of L-COMM-FACE is generally very good, indicating its use of common directions to approximate the Newton direction is very effective.
- L-COMM is an excellent approximation of L-COMM-FACE. Its convergence is similar or slightly inferior to that of L-COMM-FACE.
- In general, OWLQN has the worst convergence speed in terms of iterations.

Overall results are consistent with the analysis in Section 4. If we also consider the cost per iteration, NEWTON and L-COMM-FACE are not practical. From Table 1 each their iteration is $\mathcal{O}(m)$ times more than that of OWLQN and L-COMM, and their saving on iterations is not enough to compensate the higher cost per iteration.

We expect L-COMM to be the best in terms of running time because it needs fewer iterations than OWLQN

[4] A backtracking line search with initial step size = 1 is used.

and requires only comparable cost per iteration. This will be confirmed in Section 6.2.

6.2 Timing Comparison of Distributed Implementations. We consider a distributed implementation of OWLQN and L-COMM using OpenMPI [4]. We use 32 m4.2xlarge nodes on AWS with 1Gbps network bandwidth. The strategy in Section 3.4 is applied to possibly reduce the number of line-search steps. Since the focus is on comparing distributed implementations, we do not parallelize computation at each local node.

Results in Figure 4 show that L-COMM always converges faster than OWLQN in terms of iterations, which is consistent with the results in Section 6.1. Next we present running time also in Figure 4. In Sections 3-4 we have indicated that L-COMM is slightly more expensive than OWLQN per iteration, so the smaller number of iterations may not lead to shorter total time. However, the analysis in Section 5 shows that both methods have similar communication cost per iteration. Because in a distributed setting the communication may take a large portion of the total running time, we observe that figures for timing comparisons are very close to those for iterations. An exception is criteo, which has a relatively smaller number of features. Thus, for this set the communication cost is insignificant among the total cost. Overall, this experiment shows that the proposed L-COMM method is more efficient than OWLQN in a distributed environment.

7 Conclusions

In this work we present a limited-memory common-directions method for L1-regularized classification. The method costs slightly more than OWLQN per iteration, but may need much fewer iterations. Because both methods' communication is proportional to the number of iterations, our method is shown to be faster than OWLQN in distributed environments. In addition, we give a unified interpretation of methods for L1-regularized classification. From that we see our method is a more principled approach than OWLQN.

References

[1] G. Andrew and J. Gao. Scalable training of L1-regularized log-linear models. In *ICML*, 2007.

[2] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.*, 63:129–156, 1994.

[3] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: a library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[4] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European PVM/MPI Users' Group Meeting*, pages 97–104, 2004.

[5] P. Gong and J. Ye. A modified orthant-wise limited memory quasi-Newton method with convergence analysis. In *ICML*, 2015.

[6] C.-P. Lee, P.-W. Wang, W. Chen, and C.-J. Lin. Limited-memory common-directions method for distributed optimization and its application on empirical risk minimization. In *SDM*, 2017.

[7] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1):503–528, 1989.

[8] D. Mahajan, S. S. Keerthi, and S. Sundararajan. A distributed block coordinate descent method for training l1 regularized linear classifiers. *JMLR*, 2017.

[9] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. In *KDD*, 2013.

[10] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine learning in Apache Spark. *JMLR*, 17:1235–1241, 2016.

[11] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[12] P.-W. Wang, C.-P. Lee, and C.-J. Lin. The common directions method for regularized empirical loss minimization. Technical report, National Taiwan University, 2016.

[13] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR*, 11:3183–3234, 2010.

[14] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for l1-regularized logistic regression. *JMLR*, 13, 2012.