Supplementary Materials of “Limited-memory Common-directions Method for Distributed L1-regularized Linear Classification”

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

I Introduction
In this document, we present additional details and more experimental results.

II Derivation of the Direction Used in LBFGS
By using only the information from the last m iterations, the definition of $B_k$ in BFGS becomes the following in LBFGS.

$B_k = V_{k-1}^T V_{k-1} \cdot \ldots \cdot V_{k-m}^T V_{k-m} \cdot \ldots \cdot V_{k-1}$

By using only the information from the last m iterations, the definition of $B_k$ in BFGS becomes the following in LBFGS.

$B_k = V_{k-1}^T V_{k-1} \cdot \ldots \cdot V_{k-m}^T V_{k-m} \cdot \ldots \cdot V_{k-1}$

Note that in BFGS, $B_k$ is a fixed matrix, but in LBFGS, $B_k$ can change with k, provided its eigenvalues are bounded in a positive interval over k. A common choice is

$B_k = \frac{s_{k-1}^T u_{k-1}}{u_{k-1}^T u_{k-1}} I.$

By expanding $B_k$, $d_k$ can be efficiently obtained by $O(m)$ vector operations as shown in Algorithm II.

The overall procedure of LBFGS is summarized in Algorithm II.

III More Details of Limited-memory Common-directions Method
A sketch of the procedure for L1-regularized problems is in Algorithm III.

IV Line Search in Algorithms for L2-regularized Problems
Here we present the trick mentioned in Section 2.3 in the paper. At each line search iteration, we obtain $w^T x_i, d^T x_i, \forall i$ first, and then use $O(l)$ cost to calculate

$(w + \alpha d)^T x_i = w^T x_i + \alpha d^T x_i, \forall i.$

Algorithm I LBFGS.
1: Given $w_0$, integer $m > 0$, and $\beta, \gamma \in (0, 1)$
2: $w \leftarrow w_0$
3: for $k = 0, 1, 2, \ldots$ do
4: Calculate $\nabla f(w)$
5: Calculate the new direction

$$d \equiv -B \nabla f(w) \approx -\nabla^2 f(w)^{-1} \nabla f(w)$$

by using information of the previous m iterations (Algorithm II)
6: Calculate $\nabla f(w)^T d$
7: $\alpha \leftarrow 1, w^\text{old} \leftarrow w$
8: while true do
9: $w \leftarrow w^\text{old} + \alpha d$
10: Calculate the objective value $f(w)$ in (2.2)
11: if $f(w) - f(w^\text{old}) \leq \gamma \nabla f(w^\text{old})^T (w - w^\text{old})$ then
12: break
13: $\alpha \leftarrow \alpha \beta$
14: Update $P$ with $w - w^\text{old}$ and $\nabla f(w) - \nabla f(w^\text{old})$

Because $w^T x_i$ can be obtained from the previous iteration, $d^T x_i$ is the only $O(\#\text{nnz})$ operation needed. The line search cost is reduced from

$\#\text{line-search steps} \times O(\#\text{nnz})$

to

$1 \times O(\#\text{nnz}) + \#\text{line-search steps} \times O(l).$

This trick is not applicable to L1-regularized problems because the new point is no longer $w + \alpha d$.

V More on the Distributed Implementation
V.1 Complexity. The distributed implementation as mentioned in Section 5 is shown in Algorithm VI. Then we discuss the complexity below.

$$(2 + \#\text{line-search steps}) \times O(\#\text{nnz}) + O(lm^2) + O(mn)\frac{K}{K} + O(m^3)$$

VI More Experiments
The data sets used in this section are shown in Table I.
Algorithm II LBFGS Two-loop recursion.

1: \( q \leftarrow -\nabla f(w) \)
2: for \( i = k - 1, k - 2, \ldots, k - m \) do
3: \( \alpha_i \leftarrow s_i^T q / s_i^T u_i \)
4: \( q \leftarrow q - \alpha_i u_i \)
5: \( r \leftarrow (s_{k-1}^T u_{k-1} u_{k-1}^T) q \)
6: for \( i = k - m, k - m + 1, \ldots, k - 1 \) do
7: \( \beta_i \leftarrow u_i^T r / s_i^T u_i \)
8: \( r \leftarrow r + (\alpha_i - \beta_i) s_i \)
9: \( d \leftarrow r \)


1: while true do
2: Compute \( \nabla P f(w) \) by (2.5)
3: Solve the sub-problem (3.21)
4: Let the direction be \( d = P t \)
5: for \( j = 1, \ldots, n \) do
6: Align \( d_j \) with \( -\nabla P f(w) \) by (2.11)
7: \( \alpha \leftarrow 1, \ w_{old} \leftarrow w \)
8: while true do
9: Calculate \( w \) from \( w_{old} + \alpha d \) by (2.12)
10: if \( f(w) - f(w_{old}) \leq \gamma \nabla P f(w_{old})^T (w - w_{old}) \)
11: break
12: \( \alpha \leftarrow \alpha \beta \)
13: Update \( P \) and \( XP \)

VI.1 More Results by Using Different \( C \) Values. In Figure (I) we present more results with

\[ C = \{0.1C_{Best}, C_{Best}, 10C_{Best}\}, \]

where \( C_{Best} \) is the value to achieve the highest cross validation accuracy. The results are similar to \( C_{Best} \) presented in Section 6, but we observe that NEWTON converges slowly in larger \( C \) cases.

VI.2 More Results on Distributed Experiments. In Figure (II) we present more data sets for the distributed experiments. All settings are the same as in Section 6.

<table>
<thead>
<tr>
<th>Data set</th>
<th>#instances</th>
<th>#features</th>
<th>#nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>real-sim</td>
<td>72,309</td>
<td>20,958</td>
<td>3,709,083</td>
</tr>
<tr>
<td>rcv1t</td>
<td>677,399</td>
<td>47,226</td>
<td>49,556,258</td>
</tr>
<tr>
<td>news20</td>
<td>19,996</td>
<td>1,355,191</td>
<td>9,097,916</td>
</tr>
<tr>
<td>yahoolj</td>
<td>176,203</td>
<td>832,026</td>
<td>23,506,415</td>
</tr>
<tr>
<td>url</td>
<td>2,396,130</td>
<td>3,231,961</td>
<td>277,058,644</td>
</tr>
<tr>
<td>yahookr</td>
<td>460,554</td>
<td>3,052,939</td>
<td>156,436,656</td>
</tr>
<tr>
<td>epsilon</td>
<td>400,000</td>
<td>2,000</td>
<td>800,000,000</td>
</tr>
<tr>
<td>webspam</td>
<td>350,000</td>
<td>16,609,143</td>
<td>1,304,697,446</td>
</tr>
<tr>
<td>KDD2010-b</td>
<td>19,264,097</td>
<td>29,890,096</td>
<td>566,345,888</td>
</tr>
<tr>
<td>criteo</td>
<td>45,840,617</td>
<td>1,000,000</td>
<td>1,787,773,969</td>
</tr>
<tr>
<td>avazu-site</td>
<td>25,832,830</td>
<td>999,962</td>
<td>387,492,144</td>
</tr>
<tr>
<td>kdd2012</td>
<td>149,639,105</td>
<td>54,686,452</td>
<td>1,646,030,155</td>
</tr>
</tbody>
</table>
**Algorithm IV** A distributed implementation of OWLQN.

1. for $k = 0, 1, 2, \ldots$ do
2. Compute $\nabla^p f(w)$ by (2.5) and

$$\nabla L(w) = C \bigoplus_{r=1}^{K} (X_{j_r})^T \left[ \begin{array}{c} \vdots \\ \xi(y_i w^T x_i) \\ \vdots \end{array} \right]_{i \in J_r} \triangleright O(\#\text{nnz}/K); O(n) \text{ comm.}$$

3. Compute the search direction $d_j, r = 1, \ldots, K$ by Algorithm V \triangleright O(nm/K); O(m) \text{ comm.}
4. An allgather operation to let each node has

$$d = \begin{bmatrix} d_{j_1} \\ \vdots \\ d_{j_K} \end{bmatrix} \triangleright O(n/K) \text{ comm.}$$

5. for $j = 1, \ldots, n$ do
6. Align $d_j$ with $-\nabla^p f(w)$ by (2.11)
7. $\alpha \leftarrow 1, w^{\text{old}} \leftarrow w$
8. while true do
9. Calculate $w$ from $w^{\text{old}} + \alpha d$ by (2.12) and

$$f(w) = \|w\|_1 + C \bigoplus_{r=1}^{K} \sum_{i \in J_r} \xi(y_i w^T x_i) \triangleright O(\#\text{nnz})/K; O(1) \text{ comm.}$$

10. if $f(w) - f(w^{\text{old}}) \leq \gamma \nabla^p f(w^{\text{old}})^T (w - w^{\text{old}})$
11. break
12. $\alpha \leftarrow \alpha \beta$
13. $s_k \leftarrow w - w^{\text{old}}, u_k \leftarrow \nabla f(w) - \nabla f(w^{\text{old}})$
14. Remove 1st column of $S$ and $U$ if needed and

$$S \leftarrow \begin{bmatrix} S \\ s_k \end{bmatrix}, \quad U \leftarrow \begin{bmatrix} U \\ u_k \end{bmatrix}$$

15. $\rho_k \leftarrow \bigoplus_{r=1}^{K} (u_k)^T (s_k)_{J_r} \triangleright O(n/K); O(1) \text{ comm.}$

**Algorithm V** Distributed OWLQN Two-loop recursion

1. if $k = 0$ return $d_{j_r} \leftarrow -\nabla^p f(w)$
2. for $r = 1, \ldots, K$ do in parallel
3. $q_{j_r} \leftarrow -\nabla^p f(w)$ \triangleright $O(n/K)$
4. for $i = k - 1, k - 2, \ldots, k - m$ do
5. Calculate $\alpha_i$ by

$$\alpha_i \leftarrow \frac{\bigoplus_{r=1}^{K} (s_{i})_{j_r}}{\rho_i} \triangleright O(n/K); O(1) \text{ comm.}$$
6. for $r = 1, \ldots, K$ do in parallel
7. $q_{j_r} \leftarrow q_{j_r} - \alpha_i (u_i)_{j_r}$ \triangleright $O(n/K)$
8. Calculate

$$u^T_{k-1} u_{k-1} \leftarrow \bigoplus_{r=1}^{K} \left( (u_{k-1})^T (u_{k-1})_{j_r} \right)_{j_r} \triangleright O(n/K); O(1) \text{ comm.}$$
9. for $r = 1, \ldots, K$ do in parallel
10. $r_{j_r} \leftarrow \frac{\rho_{k-1} - u_{k-1}}{u_{k-1}} r_{j_r}$ \triangleright $O(n/K)$
11. for $i = k - m, k - m + 1, \ldots, k - 1$ do
12. Calculate $\beta_i$ by

$$\beta_i \leftarrow \frac{\bigoplus_{r=1}^{K} (u_{i})_{j_r}}{\rho_i} \triangleright O(n/K); O(1) \text{ comm.}$$
13. for $r = 1, \ldots, K$ do in parallel
14. $r_{j_r} \leftarrow r_{j_r} + (\alpha_i - \beta_i) (s_i)_{j_r}$ \triangleright $O(n/K)$

return $d_{j_r} \leftarrow r_{j_r}, r = 1, \ldots, K$
Algorithm VI: Distributed limited-memory common-directions method.

1. while true do
2.   Compute $\nabla^P f(w)$ by (2.5) and
   \[
   \nabla L(w) = C \bigoplus_{r=1}^{K} (X_{J_r,:})^T \left[ \begin{array}{c} \vdots \\ \xi'(y_i w^T x_i) \\ \vdots \end{array} \right]_{i \in J_r}.
   \]
   \[
   > O(\#nnz/K); O(n) \text{ comm.}
   \]
3.   Calculate
   \[
   X_{J_r,:} \nabla^P f(w)
   \]
   \[
   > O(\#nnz/K)
   \]
4.   Remove 1st column of $P$ and $U$ if needed and
   \[
   P_{J_r,:} \leftarrow [P_{J_r,:} \ \nabla^P f(w)]
   \]
   \[
   U_{J_r,:} \leftarrow [U_{J_r,:} \ X_{J_r,:} \nabla^P f(w)]
   \]
5.   Calculate
   \[
   (XP)^T D_w(XP) = \bigoplus_{r=1}^{K} (U_{J_r,:})^T (D_w)_{J_r,:} U_{J_r,:}
   \]
   \[
   > O(m^2/K), O(m^2) \text{ comm.}
   \]
   \[
   -P^T \nabla^P f(w) = - \bigoplus_{r=1}^{K} (P_{J_r,:})^T \nabla^P f(w)
   \]
   \[
   > O(mn/K); O(m) \text{ comm.}
   \]
6.   Solve
   \[
   ((XP)^T D_w(XP)) t = -P^T \nabla^P f(w)
   \]
   \[
   > O(m^3)
   \]
7.   Let the direction be
   \[
   d = Pt = [P_{J_1,:} t, \ldots, P_{J_K,:} t]^T
   \]
   \[
   > O(mn/K); O(n/K) \text{ comm.}
   \]
8.   for $j = 1, \ldots, n$ do
9.     Align $d_j$ with $-\nabla^p_j f(w)$ by (2.11)
10.   \[
   \alpha \leftarrow 1, \ w^{old} \leftarrow w
   \]
11.   while true do
12.     Calculate $w$ from $w^{old} + \alpha d$ by (2.12) and
13.     \[
   f(w) = \|w\|_1 + C \bigoplus_{r=1}^{K} \sum_{i \in J_r} \xi(y_i w^T x_i)
   \]
   \[
   > O(\#nnz/K); O(1) \text{ comm.}
   \]
14.     if $f(w) - f(w^{old}) \leq \gamma \nabla^P f(w^{old})^T (w - w^{old})$
15.     break
16.     $\alpha \leftarrow \alpha \beta$
17.     Remove 1st column of $P$ and $U$ if needed and
18.     \[
   P \leftarrow [P \ w - w^{old}]
   \]
19.     \[
   U_{J_r,:} \leftarrow [U_{J_r,:} \ X_{J_r,:}(w - w^{old})]
   \]
Figure (I): Comparison of different algorithms with $0.1C_{\text{best}}, C_{\text{best}}, 10C_{\text{best}}$, respectively from top to below for each data set. We show iteration versus the relative difference to the optimal value. Other settings are the same as in Figure (F).
Figure (II): Comparison of different algorithms by using 32 nodes. Upper: iterations. Lower: running time in seconds. Other settings are the same as Figure I.