Distributed Dual Iterative Hard Thresholding for Minimization of Hinge Loss over K-Sparse Constraint

Lezi Wang
152003817
lw462@cs.rutgers.edu

1. Introduction

Sparse inducing model is known to have numerous applications in machine learning problems, such as alleviating model overfitting and feature selection. A huge number of traditional methods achieve model sparsity by regularizing or constraining the goodness of fit with the ℓ_1 parameter norm [1]. However, using ℓ_1 norm as a convex relaxation tends to generate model bias, which has been shown to have inferior performance than directly using model parameter cardinality constraint in recent studies [2, 3]. This motives the surge of solving the sparsity-constrained empirical risk minimization with ℓ_2 regularization:

$$\min_{w \in R^d} F(w) \quad \text{s.t.} \quad \|w\|_0 \leq k$$
$$F(w) := \frac{1}{N}\sum_{i=1}^{N} f(w^\top x_i, y_i) + \frac{\lambda}{2}\|w\|^2 \tag{1}$$

where $D = \{x_i, y_i\}_{i=1}^N$ are training samples, $x_i \in R^d$, $y_i \in R$. $f(\cdot;\cdot)$ is a convex loss function parameterized by w. $\|w\|_0$ denotes the ℓ_0 norm of w, which equals to the number of non-zero entry in w.

Because of the non-convex and NP-hard nature, solving (1) is a non-trivial problem. The extensively studied compressed sensing based signal recovery algorithms, such as orthogonal matching pursuit and subspace pursuit, can be viewed as a special case of (1) with quadratic loss objective. More recently, a variant of Iterative Hard Thresholding (IHT) method has been proposed to solve more general convex objectives [5]. Instead of solving the problem in primal space, [4] proposes a sparse Lagrangian duality theory. Let $f^* : \mathbb{R} \to \mathbb{R}$ be the convex conjugate of f, that is $f^*(\alpha_i) = \max_{\alpha_i \in \mathcal{F}} [\alpha_i u_i - f(u_i)]$, where $\mathcal{F}$ is the feasible set of dual variable α_i. $\alpha = [\alpha_1, \alpha_2, ..., \alpha_N] \in R^N$. Under proper conditions, (1) can be equivalently reformulated into

$$\min_w \max_\alpha L(w, \alpha) \quad \text{s.t.} \quad \|w\|_0 \leq k, \ \ \forall i, \alpha_i \in \mathcal{F}$$
$$L(w, \alpha) := \frac{1}{N}\sum_{i=1}^{N} [\alpha_i w^\top x_i - f^*(\alpha_i)] + \frac{\lambda}{2}\|w\|^2 \tag{2}$$

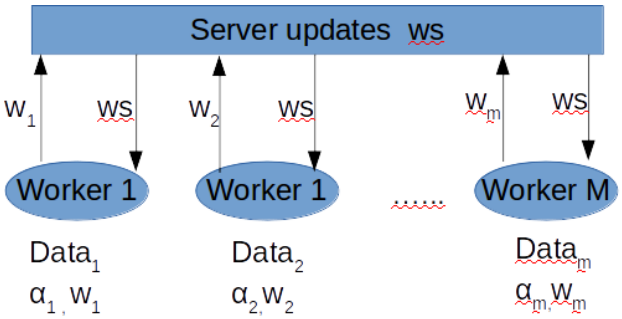

Figure 1. The framework of the proposed distributed DIHT algorithm.

Dual Iterative Hard Thresholding (DIHT) algorithms [4, 5]are proposed to solve (2). It demonstrates superior efficiency in SVM-type model training problems has been demonstrated in literature.

In this project, we propose to investigate a distributed optimization framework for DIHT algorithm. In the following context the algorithm is called as D-DIHT. The designed framework is illustrated in Fig. 1. The training set is block-wise partitioned. Training dataset $D = \{Data_1, ..., Data_m, ..., Data_M\}$ is distributed on the M workers. The worker m calculates its local optimum (w_m, α_m) and sends primal parameter update w_m to server. The server combined the sub models from workers and sends the k-sparse averaged model ws back for next update. The global minimum primal variable is obtained on server when algorithm reaches convergence.

2. D-DIHT for Distributed Non-convex Sparse Learning

Before continuing, we define some notations to be used. Let $x \in \mathbb{R}^{|}$ be a vector and F be an index set. We denote $H_F(x)$ is a truncation operator that restricts x to the set F. $H_k(x)$ is a truncation operator which preserves the top k (in magnitude) entries of x and forces the remaining to be zero.

Algorithm 1: The processing of worker m in D-DIHT algorithm.

Input : $\lambda, \eta, D_m = \{D_{m_1}, ..., D_{m_B}\}$.
Initialization $t_m = 0$.
while *no halting signal from server* **do**
(S1)Wait until receive a new ws_m from server.
(S2)$t_m = t_m + 1$.
(S3)**Initialization** $\hat{\alpha}_{[m]}^{(0)} = \alpha_{[m]}^{(t_m-1)}$,$\hat{w}_m^{(0)} = ws_m$,
$wf_m^{(0)} = \sum_{i=1}^{N_m} \alpha_{[m]_i}^{(t_m-1)} x_{[m]_i}$.
for $s = 1, 2, ...$ **do**
(SS1) Dual projected gradient ascent:
Randomly select $B^s \in \{D_{m_1}, ..., D_{m_B}\}$.
$\forall j \in B^s$

$$\hat{\alpha}_{[m]_j}^{(s)} = \mathrm{P}_{\mathcal{F}}\left(\hat{\alpha}_{[m]_j}^{(s-1)} + \eta g(\hat{\alpha}_{[m]_j}^{(s-1)}, \hat{w}_m^{(s-1)})\right), \tag{3}$$

where

$$g(\hat{\alpha}_{[m]_j}^{(s-1)}, \hat{w}_m^{(s-1)}) = x_{[m]_j}^\top \hat{w}_m^{(s-1)} - \nabla f^*(\hat{\alpha}_{[m]_j}^{(s-1)}).$$

$\forall j \notin B^s$ Set $\hat{\alpha}_{[m]_j}^{(s)} = \hat{\alpha}_{[m]_j}^{(s-1)}$.
(SS2) Primal update:

$$wf_m^{(s)} = wf_m^{(s-1)} - \frac{1}{N_m \lambda} \sum_{j=1}^{N_m} (\hat{\alpha}_{[m]_j}^{(s)} - \hat{\alpha}_{[m]_j}^{(s-1)}) x_{m_j}$$
$$\hat{w}_m^{(s)} = HT_k(wf_m^{(s)}). \tag{4}$$

end S4
$w_m^{(t_m)} = \hat{w}^{(s)}$, $\alpha_{[m]}^{(t_m)} = \hat{\alpha}_{[m]}^{(s)}$.
(S5)Send $w_m^{(t_m)}$ to server.
end

The notation $supp(x)$ represents the index set of nonzero entries of x.

Assume there are M workers and one server in the computing cluster. The training dataset D is distributed without overlap on those M workers, that is $D = \{D_1, ..., D_m, ..., D_M\}$, where $D_m = \{x_{m_i}, y_{m_i}\}_{i=1}^{N_m}$ denotes the N_m training data on worker m. The worker m can only access the data in D_m. The dual variable vector $\alpha \in R^N$ is partitioned into $\alpha = [\alpha_{[1]}, ..., \alpha_{[m]}, ..., \alpha_{[M]}]$, where $\alpha_{[m]} = [\alpha_{[m]_1}, ..., \alpha_{[m]_{N_m}}] \in R^{N_m}$ corresponds to the dual variables of D_m. The server maintains a clock value t and each worker maintains its local clock denoted as $\{t_m\}_{m=1}^M$. We present the synchronous D-DIHT algorithm in §2.1 then we extent the algorithm to asynchronous version in §**??**.

Algorithm 2: The processing of server in synchronous D-DIHT algorithm.

Input : M, k.
Initialization $w^{(0)} = 0$.
for $t = 1, 2, ...$ **do**
(S1)Wait until receive all $\{w_m^{(t)}\}_{m=1}^M$.
(S2)Average the local optimum from workers:

$$\hat{w}^{(t)} = \frac{\beta_M}{M} \sum_{m=1}^{M} w_m^{(t)}. \tag{5}$$

(S3)Hard thresholding operator

$$w^{(t)} = HT(\hat{w}^{(t)}, k). \tag{6}$$

(S4)**for** $m = 1, ..., M$ **do**
$ws = w^{(t)}$ then send ws to worker m.
end
end
Output: $w^{(t)}$.

2.1. Synchronous D-DIHT Algorithm

In the distributed computing framework shown in Fig. 1, worker machines take the major computation workload and send their local optimum primal parameter updates to server. The server machine is mainly responsible for coordinating all workers. It aggregates the primal parameter updates from workers for global optimum. We denote the primal variable that workers send to server as $\{w_m\}_{m=1}^M$ and the primal variable update sent from server to workers as $\{ws_m\}_{m=1}^M$. The processing of worker and server in synchronous mode are shown in Algorithm 1 and Algorithm 2 and elaborated as follows.

Worker: At the t-th iteration, worker $m (m = 1, ..., M)$ start updating its local primal and dual variable pair $(w_m, \alpha_{[m]})$ once receiving a new update of ws_m from server. We propose to solve the cardinality constrained ℓ_2-regularized minimization problem on local dataset $D_m = \{x_{m_j}, y_{m_j}\}_{j=1}^m$, that is

$$\min_{w_m} \max_{\alpha_{[m]}} \{\frac{1}{N_m} \sum_{j=1}^{N_m} [\alpha_{[m]_j} w_m^\top x_{m_j} - f^*(\alpha_{[m]_j})] + \frac{\lambda}{2} \|w_m\|^2\}$$
$$s.t. \ \|w\|_0 \leq k; \ \ \forall j, \ \alpha_{[m]_j} \in \mathcal{F} \tag{7}$$

(7) can be solved by DIHT algorithm outlined in (S3) and (S4) in Algorithm 1. Specifically, it iteratively updates dual variable and primal variable. The dual ascent step in (SS1) conducts dual variable ascent in mini-batch manner. $\mathrm{P}_{\mathcal{F}}$ denotes the projection operator with respect to feasible set $\mathcal{F}$. Primal variable is then updated in closed form shown

in (4), where $HT_k(v)$ is a truncation operator which preserves the top k (in magnitude) entries of v and forces the remaining to be zero. Faster local convergence on D_m can be achieved empirically by warm initialization, i.e., setting $\hat{\alpha}^0_{[m]} = \alpha^{(t_m-1)}_{[m]}, \hat{w}^0_m = ws$ in (S3).

Server: At the t-th ($t \geq 1$) iteration, the server waits until receiving all primal variable updates $\{w_m^{(t)}\}_{i=1}^M$. After that $\{w_m^{(t)}\}_{i=1}^M$ are averaged then hard-thresholding is conducted on the averaged result, which corresponds to (5) and (6) in Algorithm 2. The result $w^{(t)}$ is then sent back to all workers for next update. When the halting condition is satisfied, the server sends signal to all workers to terminate the whole optimization process.

2.2. Asynchronous D-DIHT Algorithm

The synchronous D-DIHT algorithm requires the server wait for the update from all workers. Due to many predictable and unpredictable factors such as computing power and communication delay difference of the workers, the whole-system efficiency is determined by the slowest worker. To reduce the wait time of server and faster workers we propose a asychronous D-DIHT algorithm. The workers run in the same way with synchronous setting while the server works differently with synchronous DIHT. In each iteration, the server starts updating ws once it receives the update from S workers ($S < M$).

3. Experiments

This part dedicates in showing the empirical performance of proposed D-DIHT algorithm. Eign library[1] is used for linear algebra calculation involved in all algorithms. The distributed computing is implemented by parameter server [7] which is designed as a framework for distributed machine learning problems. The testing is conducted on a cluster consisting 8 nodes interconnected by Ethernet. Each node has 2.0GB CPUs with Intel(R) Xeon(R) E5-4650 0 @ 2.70GHz. All training data are evenly distributed into worker machines without overlap.

3.1. Datasets and Hinge Loss

The binary classification dataset RCV1 is adopted for evaluation[2]. We select 480000 samples and use the provided features in dimension of 47,236.

We consider applying the proposed algorithm on minimization of the hinge loss which is widely used to train a squared SVM model [6]:

$$\min_{\|w\|_0 \leq k} \frac{1}{N} \sum_{i=1}^{N} f_{sqSVM}(y_i x_i^\top w) + \frac{\lambda}{2}\|w\|^2, \qquad (8)$$

[1] http://eigen.tuxfamily.org//index.php?title=Main_Page

[2] The dataset can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvm/

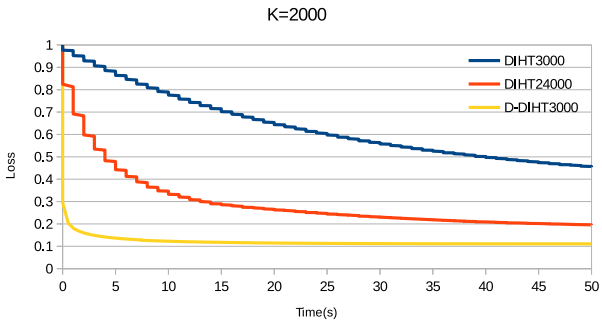

Figure 2. The convergence curve for centralized and distributed optimization in condition of $k = 2000$.

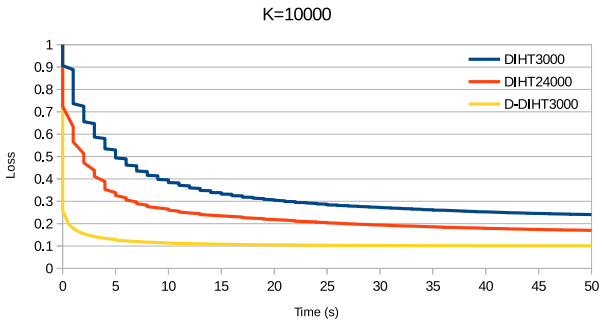

Figure 3. The convergence curve for centralized and distributed optimization in condition of $k = 10000$

where $f_{sqSVM}(y_i x_i^\top w) = \max(0, 1 - y_i x_i^\top w)^2$. The convex conjugate of $f_{sqSVM}(y_i x_i^\top w)$ is known as

$$f^*_{sqSVM}(\alpha_i) = \begin{cases} y_i\alpha_i + \frac{\alpha_i^2}{4} & \text{if } y_i\alpha_i \leq 0 \\ +\infty & \text{otherwise} \end{cases}.$$

3.2. Centralized v.s Distributed Optimization

In this part, we set up both centralized and distributed optimization on one local machine. The distributed optimization is implemented by using 8 parallel threads. The Fig. 2 and 3 shows the convergence curves for $k = 2000$ and $k = 10000$ respectively. The Y-axises in both figures indicate the loss over the whole training sets. The experiments involve three settings:

D-DIHT3000: Distributed optimization with batch size of 3000 at one iteration on each worker;

DIHT3000: Centralized optimization with batch size of 3000 at one iteration;

DIHT24000: Centralized version with batch size of 24000 (3000×8) at one iteration. Hence, the numbers of data that the centralized and distributed optimizers can utilize at every iteration are equivalent.

As shown in Fig. 2 and 3, the D-DIHT algorithm is superior than DIHT in terms of convergence rate and model

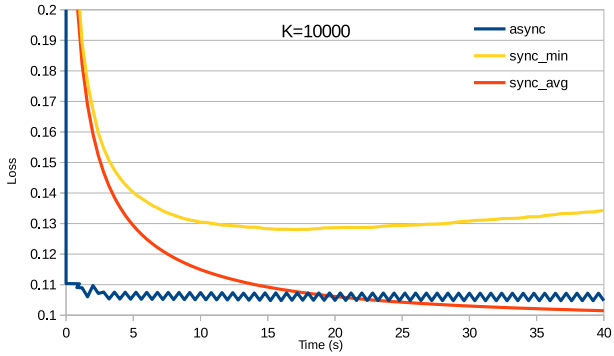

Figure 4. The convergence curves of three variants of distributed optimization when $k = 10000$.

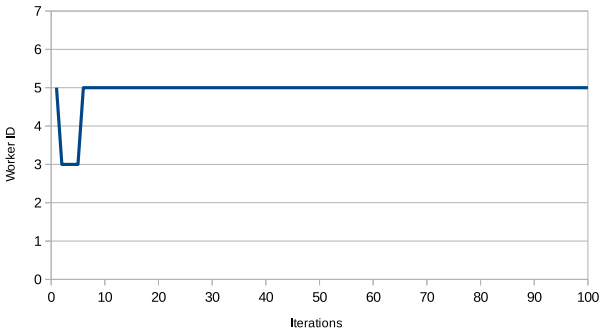

Figure 5. The worker which is picked by server if syn_min method is applied.

effectiveness. When k changes from 10000 to 2000, there are obvious degradation in performances of DIHT3000 and DIHT24000 while D-DIHT shows its robustness to the sparsity level changing.

3.3. Variants of Distributed Optimization

In this subsection, the experiments are conducted on the real case where we set up eight workers and one server through parameter server framework [7]. We aim to investigate the performance of some variants on distributed optimization:

sync_avg: Synchronous version as described in Algorithm 1 and Algorithm 2, where the IHT operation is applied to the average model on server side;

sync_min: Synchronous version where server picks the model which gives the minimum loss and applies IHK operation;

async: Asynchronous version of Algorithm 1 and Algorithm 2, where the $S = 4$ of total 8 workers.

As shown in Fig. 4, the asynchronous version achieves the fastest convergence among the three approaches and demonstrates osculations. The method of sync_avg achieves the best model effectiveness with the minimum loss at the termination. The data are evenly partitioned and stored in workers. Once each worker approaches to convergence, the computation cost on each worker can be regarded as equivalent. In our setting, workers share the same network transmission. Hence, the sequential order of sub models sent back to the server is fixed, e.g iter1: 0,1,2,3 (group1); iter2: 4,5,6,7(group2); iter3:0,1,2,3(group1); iter4: 4,5,6,7(group2)... In this case, the loss curve of $async$ demonstrates such an alternative pattern, e.g. loss for group1, loss for group2, loss for group1 ...

The loss is calculated based on the whole training data which randomly partitioned without overlap. In the strategy of $sync\_min$, it may lead to the case that up to some iterations, the model computed by the worker with the data distribution matching the validation data best will dominate the update. Fig. 5 shows the selected worker ID at each iteration on server side, where worker_5 is always selected after 5 iterations.

4. Conclusion

In this project, we proposed a D-DIHT algorithm by decentralizing the DIHT algorithm, where each worker maintains the original algorithm and server averages sub models from workers and applies IHT operation. We also explore some variants of D-DIHT.

In the future, we will investigate weighted averaging operations on server side, where the weight assigned to a given sub-model is proportional to $(regret\_of\_dataset)^p$. The $sync\_avg$ corresponds to taking $p = 0$ and $sync\_min$ basically corresponds to taking $p = infinity$ selecting only the best performing model. It was appealing that $sync\_avg$ and $sync\_min$ could be somewhat unified in this way.

References

[1] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), pages 267âĂŞ288, 1996. 1

[2] S. Bahmani, B. Raj, and P. T. Boufounos. Greedy sparsity constrained optimization. Journal of Machine Learning Research, 14(Mar):807âĂŞ841, 2013. 1

[3] C. You, D. Robinson, and R. Vidal. Scalable sparse subspace clustering by orthogonal matching pursuit. In IEEE Conference on Computer Vision and Pattern Recognition, volume 1, 2016. 1

[4] Y. Zhang and L. Xiao. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In International Conference on Machine Learning, 2015. 1

[5] X.-T. Yuan, P. Li, and T. Zhang. Gradient hard thresholding pursuit for sparsity-constrained optimization.

In International Conference on Machine Learning, 2014 1

[6] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the 21th International Conference on Machine Learning (ICML) , 2004 3

[7] Mu Li, Dave Andersen, Alex Smola, and Kai Yu. Communication Efficient Distributed Machine Learning with the Parameter Server. In Neural Information Processing Systems (NIPS), 2014 3, 4