Distributed Primal-Dual Optimization for Non-uniformly Distributed Data

Minhao Cheng[1], **Cho-Jui Hsieh** [1,2]
[1] Department of Computer Science, University of California, Davis
[2] Department of Statistics, University of California, Davis
mhcheng@ucdavis.edu, chohsieh@ucdavis.edu

Abstract

Distributed primal-dual optimization has received many focuses in the past few years. In this framework, training samples are stored in multiple machines. At each round, all the machines conduct a sequence of updates based on their local data, and then the local updates are synchronized and merged to obtain the update to the global model. All the previous approaches merge the local updates by averaging all of them with a uniform weight. However, in many real world applications data are not uniformly distributed on each machine, so the uniform weight is inadequate to capture the heterogeneity of local updates. To resolve this issue, we propose a better way to merge local updates in the primal-dual optimization framework. Instead of using a single weight for all the local updates, we develop a computational efficient algorithm to automatically choose the optimal weights for each machine. Furthermore, we propose an efficient way to estimate the duality gap of the merged update by exploiting the structure of the objective function, and this leads to an efficient line search algorithm based on the reduction of duality gap. Combining these two ideas, our algorithm is much faster and more scalable than existing methods on real world problems.

1 Introduction

Distributed optimization for large-scale machine learning has become an important research topic due to the wide availability of large datasets. In this work, we study the distributed primal-dual optimization algorithm [Ma *et al.*, 2015; Yang, 2013; Jaggi *et al.*, 2014], which has become one of the fastest algorithms for solving large-scale linear SVMs and logistic regressions. In this framework, training samples are stored distributively in multiple machines. Each machine will update the local model only based on local data, and once a while all the machines will synchronize and merge their local updates in order to obtain a new global model.

Clearly, how to merge local updates is the most crucial step of these algorithms, and several approaches have been proposed in the literature. All the current approaches use a uniform weight when merging updates: in most algorithms [Jaggi *et al.*, 2014; Yang, 2013; Ma *et al.*, 2015], local updates $\Delta \boldsymbol{w}_1, \ldots, \Delta \boldsymbol{w}_K$ on K machines are merged by $\Delta \boldsymbol{w} = s(\sum_{i=1}^{K} \Delta \boldsymbol{w}_i)$, where s is the fixed step size. [Lee and Roth, 2015] proposed a dynamic approach, showing that s can be chosen by minimizing the dual objective function.

Using a uniform weight for combining all the local updates makes sense when samples are uniformly distributed on local machines. However, when samples are non-uniformly distributed, some local updates will be more important than others, and the correlation between different data blocks can also vary hugely. In such cases, forcing the same weight will lead to slow convergence. Unfortunately, non-uniformly distributed data is very common in real world applications. For example, different data centers often contain data collected from different local areas, and it is time-consuming to randomly shuffle them before optimization. Moreover, in on-device learning problems (e.g., On-device Tensorflow), each local device only stores samples collected from a single user, which is homogeneous, and no random shuffling is allowed due to privacy issues and communication overhead, which has been discussed in recent work on federated optimization [Konečnỳ *et al.*, 2015].

In this paper, we attempt to resolve this issue by setting non-uniform weights when merging the local updates. We show that the optimal weights leading to the maximum reduction of dual objective function can be computed efficiently, and moreover there exist closed form solutions for certain functions (including square-hinge loss and hinge loss). The selection of optimal weights ensures our algorithm to make better improvement in dual objective function compared with existing approaches. Furthermore, we propose to search for the step size that directly minimizes the estimation of duality gap instead of just considering the dual objective function. Although it is computational hard to compute the duality gap exactly, we show that by sub-sampling and exploiting the structure of the L2-regularized ERM problem, we can do it in an efficient way.

Experimental results show significant improvements over existing methods when data is distributed non-uniformly. And even when data is partitioned randomly, our algorithm is still faster than all the competing methods due to the flexibility of setting K weights instead of one single weight.

2 Related Work

Large-scale ERM training is important in many real world applications, so there has been substantial amount of work developing parallel algorithms for solving this problem. Stochastic Dual Coordinate Descent (SDCA) [Shalev-Shwartz and Zhang, 2013; Hsieh *et al.*, 2008] is one of the most successful algorithms for solving ERM problems, especially for solving linear SVM. SDCA updates a dual variable at a time while maintaining the primal variable during the whole procedure.

How to parallelize SDCA in the distributed setting has been proposed in the past few years. The pioneer paper [Yang, 2013] provided an algorithm for distributed ERM training. [Jaggi *et al.*, 2014] showed that simply averaging the local updates yields good practical performance with theoretical guarantee. [Ma *et al.*, 2015] further slightly changed the local update rules to achieve faster convergence. [Lee and Roth, 2015] demonstrated that the step size could be automatically chosen by minimizing the dual objective function. [Zheng *et al.*, 2017] proposed a distributed ADMM method to solve the problem. They show their DADM algorithm is equivalent to CoCoA+. All the existing algorithms are based on merging local updates with the same weight, while we propose a novel approach to automatically select non-uniform weights for local updates when merging them. Although [Jaggi *et al.*, 2014] stated the per-block weight B_k/K in their algorithm framework, they did not provide a way to select the per-block step sizes, and it is impossible to hand-tune these weights. Therefore, they fix all the $B_k = 1$ in both of their proof and experiments. In comparison, our approach can choose per-block weight automatically, which results in a significant speed up.

Also, there are a lot of papers focusing on how to pararllelize distributively in loss function directly(mainly on SGD). For centralized SGD, a lot of papers have been published in both algorithm and implementation [Agarwal and Duchi, 2011; Dekel *et al.*, 2012; Chen *et al.*, 2016]. If there is no center node, [Konečnỳ *et al.*, 2016] compared variance reduction SGD methods in the on-devices setting with dual methods cocoA and cocoA+. [McMahan *et al.*, 2016] extended its framework into deep learning setting. [Lian *et al.*, 2017] showed decentralized SGD could perform better than centralized one with lower communication cost.

3 Setup

Given a set of training samples $\{(\boldsymbol{x}_i)\}_{i=1}^n$, $\boldsymbol{x}_i \in \mathbb{R}^d$, we consider the following L2-regularized empirical risk minimization (L2-ERM) problem:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \big\{ \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^n \ell_i(\boldsymbol{w}^T\boldsymbol{x}_i)\big\} := P(\boldsymbol{w}), \quad (1)$$

where $\ell_i(\cdot)$ are convex loss functions which may depend on the label of i-th training sample, and the constant $\lambda > 0$ is the regularization parameter. The label of testing data can then be predicted by $\text{sign}(\boldsymbol{w}^T\boldsymbol{x})$. Many machine learning models are in this form, such as linear SVM and logistic regression.

We focus on the dual form of eq (1):

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \big\{ -\frac{1}{n}\sum_{j=1}^n \ell_j^*(-\alpha_j) - \frac{\lambda}{2}\|\frac{X\boldsymbol{\alpha}}{\lambda n}\|^2\big\} := D(\boldsymbol{\alpha}), \quad (2)$$

where $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d\times n}$ is the data matrix and $\ell_j^*(\cdot)$ is the conjugate function of $\ell_j(\cdot)$. If we define

$$\boldsymbol{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda n} X\boldsymbol{\alpha}, \quad (3)$$

then the duality gap is given by

$$G(\boldsymbol{\alpha}) = P(\boldsymbol{w}(\boldsymbol{\alpha})) - D(\boldsymbol{\alpha}), \quad (4)$$

and based on the primal-dual relationship we know $G(\boldsymbol{\alpha}^*) = 0$ where $\boldsymbol{\alpha}^*$ is the dual optimal solution.

Distributed primal-dual optimization is one of the most efficient ways for solving L2-ERM problems. Instead of solving primal problem, they solve the dual problem while maintaining and communicating on the corresponding primal solutions. Consider the distributed system with K machines and each machine has a subset of training samples. We then partition dual variables by the same way with their associated training samples, and use $\{S_k\}_{k=1}^K$ to denote this non-overlapping index partition. We use $\pi(i)$ to denote the partition that the i-th index belongs to, and for any vector $\boldsymbol{\alpha} \in \mathbb{R}^n$ we define $\boldsymbol{\alpha}_{[k]}$ to be the vector that

$$(\boldsymbol{\alpha}_{[k]})_i = \begin{cases} 0 & \text{if } i \notin S_k \\ \alpha_i & \text{otherwise.} \end{cases}$$

Assume $\boldsymbol{\alpha}^t$ and $\boldsymbol{w}^t = \boldsymbol{w}(\boldsymbol{\alpha}^t)$ are the solution after the t-th synchronization step. Then at the current iteration all the K machines perform updates in parallel by solving the subproblem associated with local training samples:

$$\arg\min_{\Delta\boldsymbol{\alpha}_{[k]}} \big\{D(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}_{[k]})\big\} := -\frac{1}{n}\sum_{i\in S_k} \ell_i^*(-\alpha_i - (\Delta\boldsymbol{\alpha}_{[k]})_i) - \frac{1}{n^2}\boldsymbol{w}^T X\Delta\boldsymbol{\alpha}_{[k]} - \frac{\lambda}{2}\|\frac{1}{\lambda n}X\Delta\boldsymbol{\alpha}_{[k]}\|^2. \quad (5)$$

And then after a while the local updates are synchronized to obtain the global solution at step $t+1$:

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t + \eta\sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\ ,\ \boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta\sum_{k=1}^K \Delta\boldsymbol{w}_{[k]}. \quad (6)$$

where the step size η can be viewed as a weight for merging all the local updates. CoCoA [Jaggi *et al.*, 2014] used a fixed step size, and show that $\eta = 1/K$ guarantees the convergence. [Yang, 2013; Ma *et al.*, 2015] also used a fixed step size, but proposed slight different ways to form the subproblem (5). BQO [Lee and Roth, 2015] shows that η could be computed by minimizing the dual objective function when ℓ_i^* is simple (e.g., SVM). This primal-dual distributed optimization framework is presented in Algorithm 1. In this paper, we propose a more flexible way to combine local models.

4 Non-uniform Weights for Merging Local Updates

As described in the previous section, all the existing algorithms select a single weight η to obtain the new solution; however, data is often non-uniformly distributed in local machines, and each local update should have different weights contributing to the global model. This heterogeneity was not taken into consideration in existing approaches.

Algorithm 1: Distributed Primal-Dual Optimization Framework

Input : Training data X, index partition $\{S_k\}_{k=1}^K$, initial solution $\boldsymbol{\alpha}^0$ and $\boldsymbol{w}^0 = \boldsymbol{w}(\boldsymbol{\alpha}^0)$
Output: Primal solution $\boldsymbol{\alpha}^*$ and dual solution $\boldsymbol{w}^*$.

for $t = 0, 1, \ldots$ **do**
- Obtain an approximate solution $\Delta\boldsymbol{\alpha}_{[k]}$ of the local subproblem (5) **for each node** k.
- Choose a step size η.
- Update $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta \sum_{k=1}^K \Delta\boldsymbol{w}_{[k]}$.
- Update $\boldsymbol{\alpha}_{[k]}^{(t+1)} \leftarrow \boldsymbol{\alpha}_{[k]}^t + \eta\Delta\boldsymbol{\alpha}_{[k]}$ **for each node** k.

Based on this observation, we propose to have different weights for different blocks. More specifically, after computing local updates $\Delta\boldsymbol{\alpha}_{[k]}$ and $\Delta\boldsymbol{w}_{[k]}$ for all the machines, we update the global solution by the following weighted average:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \sum\nolimits_{k=1}^K \beta_k \Delta\boldsymbol{w}_{[k]}, \tag{7}$$

where $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \cdots\ \beta_K]^T$ are the weights for local updates. The update rule used in all the previous work (6) is a special case of (7) when $\beta_1 = \beta_2 = \cdots = \beta_K = \eta$.

It is impossible to hand-tune all these weights. To overcome this issue, we propose an efficient way to compute the optimal weights $\{\beta_i\}_{i=1}^K$ by maximizing dual objective function, with small communication overhead. The dual objective function after the update can be written as

$$D(\boldsymbol{\alpha} + \sum_{k=1}^K \beta_k \Delta\boldsymbol{\alpha}_k) = -\frac{1}{n}\sum_{k=1}^K \sum_{i\in S_k} \ell_i^*(-\alpha_i - \beta_i \Delta\boldsymbol{\alpha}_i) - \boldsymbol{p}^T\boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^T M\boldsymbol{\beta} + \text{constant} \tag{8}$$

where X is the $\mathbb{R}^{d\times n}$ data matrix, M is a K by K matrix with $M_{sq} = \frac{1}{\lambda n^2}\Delta\boldsymbol{w}_{[s]}^T\Delta\boldsymbol{w}_{[q]}$ and $p_k = \frac{1}{\lambda n^2}\boldsymbol{w}^T\Delta\boldsymbol{w}_{[k]}$. Therefore, the optimal $\boldsymbol{\beta}$ can be chosen by solving the following k-variate optimization problem:

$$\boldsymbol{\beta}^* = \operatorname*{argmin}_{\boldsymbol{\beta}} -D(\boldsymbol{\alpha} + \sum_{k=1}^K \beta_k \Delta\boldsymbol{\alpha}_{[k]}) \tag{9}$$
$$= \operatorname*{argmin}_{\boldsymbol{\beta}} \frac{1}{2}\boldsymbol{\beta}^T M\boldsymbol{\beta} + \boldsymbol{p}^T\boldsymbol{\beta} + \sum_{i=1}^n \ell_i^*(-\alpha_i - \beta_{\pi(i)}\Delta\alpha_i),$$

where $\Delta\alpha_i = (\Delta\boldsymbol{\alpha}_{[\pi(i)]})_i$ is the change of the i-th dual variable.

How to solve this small K-variate subproblem distributedly? We first consider a set of simple but useful functions, where ℓ_i^* is just a simple linear or quadratic function with bounded constraints. More specifically,

$$\ell_i^*(\alpha) = a\alpha + b\alpha^2 + I(\alpha \in [C, D]), \tag{10}$$

where $I(\alpha \in [C, D])$ taking value 0 when $\alpha \in [C, D]$ and ∞ outside the constraint. Most of the widely used applications follow this framework. For example, in SVM $\ell_i = \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)$ (hinge loss), and the conjugate loss is $\ell_i^*(\alpha) = \alpha y_i I(y_i\alpha \in [-1, 0])$, where $I(\alpha) = 1$ if $\alpha \in [0, y_i]$ and is infinity when α is outside the range. For L2-SVM, $\ell_i = \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2$, and the conjugate is $\ell_i^*(\alpha) = (\alpha^2 + 4\alpha)I(y_i\alpha \le 0)$. In these cases, (9) can be written as

$$\boldsymbol{\beta}^* = \operatorname*{argmin}_{\boldsymbol{\beta}} \frac{1}{2}\boldsymbol{\beta}^T M\boldsymbol{\beta} + \bar{\boldsymbol{p}}^T\boldsymbol{\beta} \text{ s.t. } \bar{C} \le \boldsymbol{\beta} \le \bar{D}, \tag{11}$$

where $\bar{C}, \bar{D}, \bar{\boldsymbol{p}}$ can be easily obtained by substituting (10) into (9). After forming M and $\bar{\boldsymbol{p}}$, the subproblem (11) can be solved in around $O(K^3)$ time using classical constrained minimization approaches. K is number of nodes, which is typically small.

For a general loss function (e.g., logistic regression), we can conduct gradient descent to solve (9). At each iteration, the main difficulty is to compute the third term, so each machine has to compute the scalar $\sum_{i\in S_k} \ell_i^{*'}(-\alpha_i - \beta_k\Delta\alpha_i)$ and then communicates with others. This only requires $O(1)$ communication and $O(n/K)$ computation, so will be quite efficient in a communication-dominated environment. Basically the cost is the same with one of the previous approaches [Lee and Roth, 2015].

How to obtain $M, \bar{\boldsymbol{p}}$? The next question is how to form M and $\bar{\boldsymbol{p}}$. $\bar{p}_k$ is mainly related to $\boldsymbol{w}^T\Delta\boldsymbol{w}_{[k]}$, which can be computed in $O(d)$ time and aggregated to a master machine with $O(1)$ communication. Thus, there is almost no overhead for getting this term. For computing M, we mainly need to distribute each $\Delta\boldsymbol{w}_{[k]}$ to other machines, and each machine takes $O(dK)$ time to compute a column of M. This operation requires $O(dK)$ communication, which is more expensive then the original $O(d)$ communication cost.

However, we observe that $M = \Delta W^T\Delta W$ and $\Delta W = [\Delta\boldsymbol{w}_{[1]}\ \Delta\boldsymbol{w}_{[2]}\ \cdots\ \Delta\boldsymbol{w}_{[K]}]$ is a d by K matrix with $d \gg K$. Therefore, we can use either sampling or sketching technique to reduce the communication cost, while still maintain a good enough approximation of M in order to compute the step size. For example, if we use Gaussian sketching, the matrix is approximated by

$$\bar{M} = \Delta W^T S^T S\Delta W, \tag{12}$$

where S is a matrix whose rows are i.i.d Gaussian random vectors. Here $S \in \mathbb{R}^{q\times d}$ and we hope $d \gg q$ because each machine can compute the sketching on local variables $S\Delta\boldsymbol{w}_{[i]}$ and then use $O(q)$ communication to form $\bar{M}$. In Theorem 1, we will show that the algorithm is guaranteed to linearly converge with high probability when $q = O(K\log K)$.

Using this approach, the communication cost can be reduced to $O(qK)$. Moreover, our step size selection strategy can be used for any set of local updates $\Delta\boldsymbol{w}_{[k]}$, so we can also combine with the modified local subproblem used in [Ma *et al.*, 2015; Yang, 2013].

Next, we discuss the theoretical benefit of our proposed approach. Since our step size $\boldsymbol{\beta}^*$ maximizes the dual objective function, it has the following nice property:

Lemma 1. *The block-wise step sizes $\boldsymbol{\beta}^*$ computed by solving* (9) *is guaranteed to reduce the objective function more than any single constant step size:*

$$D(\boldsymbol{\alpha} + \sum\nolimits_{k=1}^K \beta_k^*\Delta\boldsymbol{\alpha}_{[k]}) \ge D(\boldsymbol{\alpha} + \eta\sum\nolimits_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}) \quad \forall\eta \in \mathbb{R}.$$

As a result, our step size selection is guaranteed to be better than previous work. Moreover, we further discuss how the convergence will be affected by using the sampling technique to compute M in the following Theorem.

Theorem 1. *If we approximate M by the sketched matrix $\bar{M}$ defined in* (12) *with $q = O(K \log K)$, then with high probability, $\boldsymbol{\beta}^*$ obtained by solving the approximate system* (9) *will satisfy*

$$D(\boldsymbol{\alpha} + \sum_{k=1}^{K} \beta_k \Delta\boldsymbol{\alpha}_{[k]}) - D(\boldsymbol{\alpha}) \geq$$

$$\frac{\varepsilon}{2}(D(\boldsymbol{\alpha} + \eta \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]}) - D(\boldsymbol{\alpha})) \quad \forall \eta \in \mathbb{R}.$$

Proof: Let $\bar{f}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^T \bar{M} \boldsymbol{\beta} + \boldsymbol{p}^T \boldsymbol{\beta} + \sum_{i=1}^{n} \ell_i^*(-\alpha_i - \beta_{\pi(i)} \Delta\alpha_i)$ and $\bar{\boldsymbol{\beta}} = \operatorname{argmin}_{\boldsymbol{\beta}} \bar{f}(\boldsymbol{\beta})$. We can easily get $\bar{f}(\bar{\boldsymbol{\beta}}) \leq \bar{f}(\boldsymbol{\beta}^*) = f(\boldsymbol{\beta}^*) + \frac{1}{2}\boldsymbol{\beta}^{*T}(M - \bar{M})\boldsymbol{\beta}^*$. From [Tropp and others, 2015], we can bound $\boldsymbol{\beta}^{*T}(M - \bar{M})\boldsymbol{\beta}^*$ with $||\boldsymbol{\beta}^*||_M^2$. Since the first term of $f(\cdot)$ is quadratic and $\ell_i^*(-\alpha_i - \beta_{\pi(i)}\Delta\alpha_i)$ is convex, $\bar{f}(\boldsymbol{\beta}) \leq f(\boldsymbol{\beta}^*) - \frac{\varepsilon}{2} f(\boldsymbol{\beta}^*)$. □

Therefore, we are guaranteed to make more progress than the right-hand side, which is the best objective function reduction that can be achieved by previous approaches with one single weight.

Based on Theorem 1, we can use the same analysis in [Ma *et al.*, 2015] to show the convergence rate of our algorithm. Assume each local solver achieves the "Θ approximate solution", which means

$$E[D(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}^*_{[k]}) - D(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}_{[k]})] \leq \Theta\big(D(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}^*_{[k]}) - D(\boldsymbol{\alpha})\big) \tag{13}$$

where $\Delta\boldsymbol{\alpha}^*_{[k]}$ is the optimal solution of (5). We then have the following results: If ℓ_i is $1/\mu$-smooth for all i, then after

$$T \geq \frac{1}{K(1-\Theta)} \frac{\lambda\mu n + \sigma_{\max}}{\lambda\mu n} \log(1/\epsilon_D)$$

iterations we have $E[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)] \leq \epsilon_D$. And after

$$T \geq \frac{K(\lambda\mu n + \sigma_{\max})}{(1-\Theta)\lambda\mu n} \log\big(\frac{\lambda\mu n + \sigma_{\max}}{\gamma(1-\Theta)\lambda\mu n} \frac{1}{\epsilon_G}\big)$$

iterations we have $E[G(\boldsymbol{\alpha}^T)] \leq \epsilon_G$, where we assume $\|\boldsymbol{x}_i\| \leq 1$ for all training samples and $\sigma_{\max} = \max_k |S_k|$. Similarly, our algorithm has the same guarantee with CoCoA when ℓ_i is Lipschitz continuous (Theorem 8 of [Ma *et al.*, 2015]).

However, we observe that if we only use this approach, the duality gap, although smaller than COCOA, will oscillate and is not stable. The main reason is that we highly optimize the dual objective function but do not consider primal function value in this approach. We then tackle this question in the next section.

5 Proposed Approach II: Stochastic Line Search on Duality Gap

Although our algorithm is faster than existing approaches in terms of achieving a lower dual objective function, this does not imply a better duality gap. The same phenomenon has been also observed for the BQO [Lee and Roth, 2015] approach which finds a single step size by minimizing the dual objective function. Motivated by this observation, we further propose a line search algorithm to minimize an estimator of duality gap. Combined with the BlockLS approach, we are able to achieve improvement in terms of both duality gap and primal objective function value.

The duality gap (4) is hard to compute exactly. However, we show that an estimation of duality gap can be computed efficiently. We define an approximate duality gap by

$$\tilde{G}(\boldsymbol{\alpha}) = \tilde{P}(\boldsymbol{w}(\boldsymbol{\alpha})) - D(\boldsymbol{\alpha}),$$
$$\text{where} \quad \tilde{P}(\boldsymbol{w}(\boldsymbol{\alpha})) = \frac{1}{|U|}\sum_{i \in U} \ell_i(\boldsymbol{w}^T \boldsymbol{x}_i) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2, \tag{14}$$

where U is a subset sampled from $\{1, 2, \cdots, n\}$. We then approximate the primal function by sub-sampling, and keep the exact dual objective function value. Typically $|U|$ can be very small comparing to the full dataset, and we set $|U| = 0.01n$ in all the experimental results.

Given the BQO step size $\boldsymbol{\beta}$ and the local updates $\Delta\boldsymbol{w}_{[k]}, \Delta\boldsymbol{\alpha}_{[k]}$ for all $k = 1, \cdots, K$, the approximate duality gap can be computed efficiently by the following approach. First, the dual objective function can be computed by

$$D(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \beta_k \Delta\boldsymbol{\alpha}_{[k]}) = D(\boldsymbol{\alpha}) - (\boldsymbol{p}^T\boldsymbol{\beta})\gamma - (\frac{1}{2}\boldsymbol{\beta}^T M \boldsymbol{\beta})\gamma^2$$
$$+ \frac{1}{n}\sum_{j=1}^{n}(\ell_j^*(-\alpha_j) - \ell_j^*(-\alpha_j + \gamma\beta_{\pi(j)}\Delta\alpha_j)). \tag{15}$$

As discussed in the previous section, in many applications (SVM, L2-SVM or square loss) the last term is just a linear term with bounded constraints that can be computed efficiently. The second and third term have already been computed in BlockLS. So in these cases (15) can be computed in $O(1)$ time. If we consider a general loss function, then similar to the previous section, we need $O(1)$ more communication to gather the last term.

The approximate primal function $\tilde{P}$ can be computed by

$$\tilde{P}(\boldsymbol{w} + \gamma \sum_{k=1}^{K} \beta_k \Delta\boldsymbol{w}_{[k]}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + (\lambda^2 n^2 \boldsymbol{p}^T\boldsymbol{\beta})\gamma +$$
$$(\frac{n^2}{2}\boldsymbol{\beta}^T M \boldsymbol{\beta})\gamma^2 + \frac{1}{|U|}\sum_{i=1}^{n} \ell_i(\boldsymbol{w}^T\boldsymbol{x}_i + \gamma(\sum_{k=1}^{K} \beta_k \Delta\boldsymbol{w}_{[k]\boldsymbol{x}_i}^T)). \tag{16}$$

Again, we already have $\boldsymbol{p}$ and M, so the main difficulty is to compute the final term in (16). Unfortunately, ℓ_i are often quite complex. Examples include hinge loss (SVM) and square-hinge loss (L2-SVM). This is also the reason that we need to approximate the primal function instead of computing it exactly. Before line search for γ, we will pre-compute $\boldsymbol{w}^T\boldsymbol{x}_i$ and $\sum_{k=1}^{K} \beta_k \Delta\boldsymbol{w}_{[k]}^T \boldsymbol{x}_i$, where the second one can be computed in each node and then synchronize together with others. As a result, we need $O(|U|\bar{d})$ time for pre-computing those values, where $\bar{d}$ is average number of nonzero features

Dataset	Training Size	Features(d)	Sparsity	λ	Worker
epsilon	400,000	2,000	100%	1	32
rcv1	677,399	47,236	0.16%	1	32
covtype	522,911	54	22.22%	0.001	8

Table 1: Data statistics

for each training sample. After pre-computing these two values, we just need $O(|U|)$ time to compute (16) for each γ.

In summary, in the line search procedure for γ, we will first need $O(|U|\bar{d})$ time to pre-compute, and after that each iteration we only need $O(|U|)$ time to compute the approximate duality gap $\tilde{G}(\boldsymbol{\alpha} + \gamma(\sum_{k=1}^{K} \beta_k \Delta\boldsymbol{\alpha}_{[k]}))$.

Using this approach, we can approximately compute the duality gap efficiently for each choice of step size γ. We propose to have a backtracking step size selection rule until

$$\tilde{G}(\boldsymbol{\alpha} + \gamma(\sum_{k=1}^{K} \beta_k \Delta\boldsymbol{\alpha}_{[k]})) \leq \sigma\tilde{G}(\boldsymbol{\alpha} + \frac{1}{K}(\sum_{k=1}^{K} \beta_k \Delta\boldsymbol{\alpha}_{[k]}))).$$

And we force $\gamma \geq 1/K$ for this line search procedure, which ensures the convergence of the algorithm if ℓ^* satisfies (10). To prove this, since $D(\cdot)$ is a quadratic function on γ and after line search $\gamma \in [1/K,\ 1]$, we have $D(\boldsymbol{\alpha} + \gamma(\sum_k \beta_k \Delta\boldsymbol{\alpha}_{[k]}))$ is larger than the dual objective function value of the right-hand side, which implies a sufficient decrease of dual objective function. Thus, the convergence is guaranteed.

6 Experimental Results

In this section, we combine the Block-wise step size selection algorithm with Stochastic line search and propose it as "Stochastic BlockLS" algorithm. We use SDCA as our local solver and apply each method to the binary hinge-loss support vector machines. We include the following algorithms into comparison:

- Stochastic BlockLS: our proposed algorithm, using 1% of training samples to conduct stochastic line search on duality gap, and $q = 10^{-2}n$ for approximating M.
- CoCoA [Jaggi *et al.*, 2014]: they set the step size $\eta = 1/K$.
- CoCoA+ [Ma *et al.*, 2015]: they slightly changed the local subproblems and used a constant step size to synchronize the updates..
- BQO [Lee and Roth, 2015]: conducted an exact line search on dual objective function. We set $\tau = 0.001$ as suggested in the original paper.

[Zheng *et al.*, 2017] proved their DADM algorithm equivalent to CoCoA+ and their performance improvement was got by using acceleration which could be used in our algorithm as well. Therefore, we don't include it in our experiment.

Datasets and Settings. We consider three datasets: rcv1, epsilon, covtype. Details are shown in Table 1. Since epsilon and covtype doesn't have the testing dataset. We split 87.5% data instances to training set and 12.5% to testing set. To have a fair comparison between all algorithms, we implement all methods in C++ using MPI-LIBLINEAR setting.

In our experiments, we use 8-32 nodes in the TACC Stampede cluster where each node is with 256GB memory. Instances are split into our nodes in a uniform random style in covtype and rcv1. The comparison results are shown in Figure 2. We test the algorithms using the default λ values used in previous papers, as listed in the Table 1. Also, results with different λ values are showed in the as well.

Uniform vs non-uniform distributed data: to verify our algorithm works well for the non-uniformly distributed setting, we do K-Means clustering on epsilon and distribute different clusters to different nodes. This is to simulate the case when data is non-uniformly distributed. The comparison between uniformed and non-uniformed dataset is shown in Figure 3.

Results. We show the results on both epoch and time versus duality gap. In all datasets, Stochastic BlockLS algorithm is faster than CoCoA, CoCoA+ and BQO. In Figure 2, our method outperforms other methods in both duality gap and primal function value. Also, our algorithm is much more stable than BQO. In Figure 3, our algorithm performs better in non-uniformly distributed datasets, where every node should have a different weight because of the heterogeneous property. To be noted, although duality gaps are in 10^2, however, all relative duality gaps are around 10^{-3}, which will generate an approximate optimal solution.

Sensitivity to parameters. We also test our algorithm on rcv1 with different choices of regularization parameters. Due to the page limit, we can't show the results here. However, We find that our algorithm is consistently better for $\lambda = 0.1, 10, 100$; BQO is good for small λ but becomes extremely bad for large λ, and CoCoA/CoCoA+ are always much slower than our method.

Scalability. Figure 1 shows the speedup versus number of nodes. We observe that our algorithm has a better scalability than CoCoA and CoCoA+. The speedup is calculated by the time needed to obtain the same primal function value divided by the time to get the same primal function value in 2 machine. As number of machines increases, data distribution becomes more unbalanced. Therefore, our method, which calculates each block's step-size differently instead of doing a simple average, could have a better performance with large number of nodes.

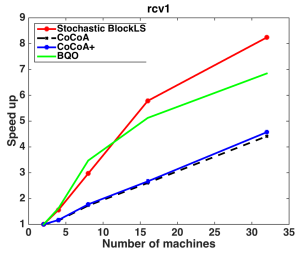


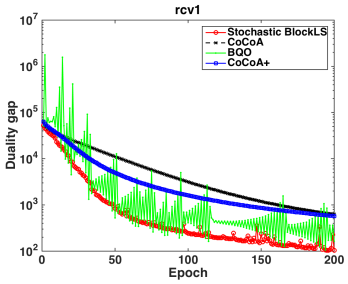
Figure 1: Scaling up with number of nodes

7 Conclusions.

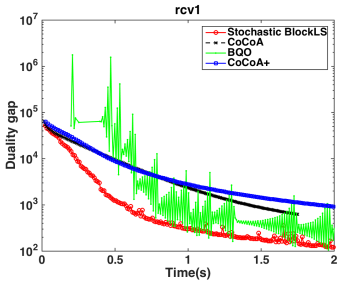
In this paper, we propose a novel line search approach for both uniformly and non-uniformly distributed primal-dual optimization framework.

Acknowledgments

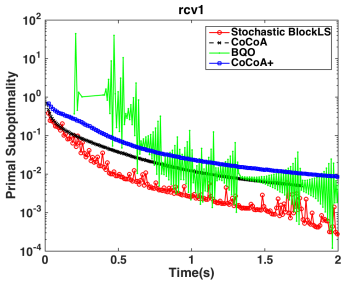
Cho-Jui Hsieh and Minhao Cheng acknowledge the support of NSF via IIS-1719097



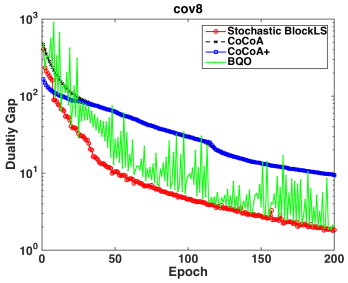
(a) Duality gap versus epoch



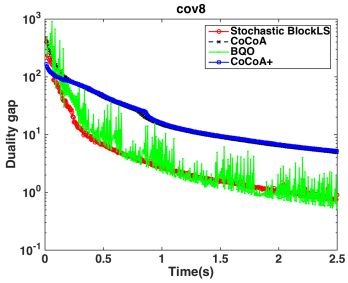
(b) Duality gap versus time



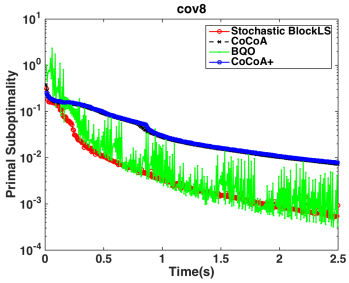
(c) Relative Primal function value versus time



(d) Duality gap versus epoch on covtype

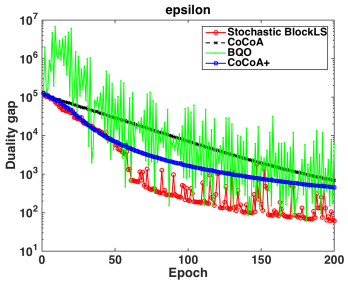


(e) Duality gap versus time on covtype

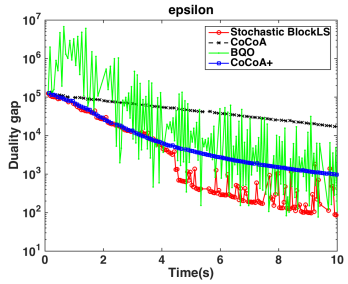


(f) Relative Primal function value versus time on covtype

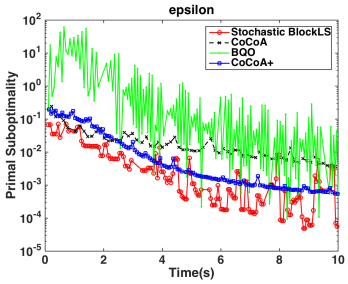
Figure 2: Comparison on duality gap and relative primal function value on uniformly distributed datasets.



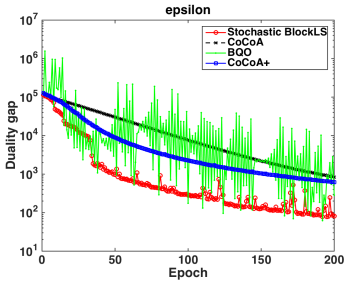
(a) Duality gap comparison versus epoch on uniformed epsilon



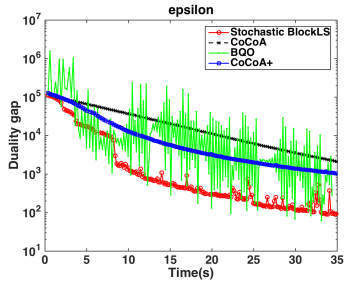
(b) Duality gap comparison versus time on uniformed epsilon



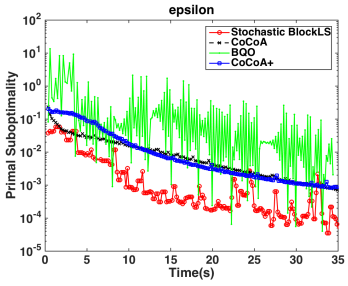
(c) Relative Primal function value comparison versus time on uniformed epsilon



(d) Duality gap comparison versus epoch on non-uniformed epsilon



(e) Duality gap comparison versus time on non-uniformed epsilon



(f) Relative Primal function value comparison versus time on non-uniformed epsilon

Figure 3: Comparison on duality gap and relative primal function value on non-uniformly distributed datasets.

References

[Agarwal and Duchi, 2011] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[Chen *et al.*, 2016] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[Dekel *et al.*, 2012] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[Hsieh *et al.*, 2008] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008.

[Jaggi *et al.*, 2014] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *NIPS*. 2014.

[Konečnỳ *et al.*, 2015] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[Konečnỳ *et al.*, 2016] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[Lee and Roth, 2015] C.-P. Lee and D. Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *ICML*, 2015.

[Lian *et al.*, 2017] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5336–5346, 2017.

[Ma *et al.*, 2015] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. averaging in distributed primal-dual optimization. In *ICML*, 2015.

[McMahan *et al.*, 2016] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[Shalev-Shwartz and Zhang, 2013] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.

[Tropp and others, 2015] Joel A Tropp et al. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

[Yang, 2013] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, 2013.

[Zheng *et al.*, 2017] Shun Zheng, Jialei Wang, Fen Xia, Wei Xu, and Tong Zhang. A general distributed dual coordinate optimization framework for regularized loss minimization. *Journal of Machine Learning Research*, 18(115):1–52, 2017.