2016 IEEE International Conference on Data Science and Advanced Analytics

Efficient Sampling-based ADMM for Distributed Data

Jun-Kun Wang
National Taiwan University
wangjim123@gmail.com

Shou-De Lin
National Taiwan University
sdlin@csie.ntu.edu.tw

***Abstract*—This paper presents two strategies to speed up the alternating direction method of multipliers (ADMM) for distributed data. In the first method, inspired by stochastic gradient descent, each machine uses only a subset of its data at the first few iterations, speeding up those iterations. A key result is in proving that despite this approximation, our method enjoys the same convergence rate in terms of the number of iterations as the standard ADMM, and hence is faster overall. The second method also follows the idea of sampling a subset of the data to update the model before the communication of each round. It converts an objective to the approximated dual form and performs ADMM on the dual. The method turns out to be a distributed variant of the recently proposed SDCA-ADMM. Yet, compared to the straightforward distributed implementation of SDCA-ADMM, the proposed method enjoys less frequent communication between machines, better memory usage, and lighter computational demand. Experiments demonstrate the effectiveness of our two strategies.**

I. Introduction

We consider training a classifier using a large, distributed data set. Because the data are distributed across different machines and are too big to relocate, a *distributed* optimization model is the most efficient means to train a classifier. The *alternating direction method of multipliers (ADMM)*, in which each local machine updates its learned model and a master machine tries to reach a consensus, has been shown to be a particularly effective method for distributed optimization [5], [24]. We consider a setting with g machines such that each machine m learns a model w_m from its local data N_m. The learned models w_m are required to agree with each other to form a consensus c. Mathematically, we are interested in solving the following linear classification problem,

$$\begin{aligned} \underset{w_1,\ldots,w_g,c}{\text{minimize}} \; & \sum_{m=1}^{g} \sum_{i\in N_m} C \, \max(1 - l_i w_m^\top z_i, 0)^2 + \frac{1}{2}\|c\|_2^2 \\ \text{s.t. } & w_m - c = 0, \forall j, \end{aligned} \tag{1}$$

where $(l_i \in \mathbb{R}, z_i \in \mathbb{R}^p)$ is a label/feature pair of the i_{th} sample on machine m, and C controls the regularization effect. Although squared hinge loss is shown here, the objective is in fact general and other types of loss function can be utilized. For example, if squared loss is chosen, the objective becomes a regression problem.

Prior work on using ADMM for distributed data observed that the update of w_m at each iteration is similar to the standard SVM formulation, and hence stochastic dual coordinate ascent [10] could be adapted to solve ADMM for distributed data [24]. In this approach, however, several "inner" iterations are required to obtain w_m at each iteration of ADMM. When the data on each local machine are large, the computational cost for one pass over the full local data set drastically increases, making the update of w_m quite expensive. In this paper, we propose two sampling-based methods to speed up ADMM for large distributed data.

Our first method is inspired by the use of sampling in stochastic gradient descent (SGD) [3], an effective method for solving large-scale optimization problems. At each iteration, instead of using *all* the data as in traditional gradient descent, SGD *samples* an instance to compute the gradient. It is known that only a few samples are needed for SGD to achieve sufficient descent of the objective at the beginning. To achieve a similar effect in ADMM, we propose that at the first few iterations, each local machine uses only a subset of its data (instead of using all its data) to update its model. For example, at the first iteration, each machine m samples half of its data to compute w_m. Then, the sample size is increased for the next iteration, and eventually each machine utilizes all of its local data. The method requires fewer computations at the first few iterations, and therefore can achieve faster convergence, due to cheaper iteration costs at the beginning. More importantly, we provide a theorem to show that the method enjoys the same convergence rate in terms of the number of the iterations as the standard ADMM.

Our second sampling-based method for speeding up the use of ADMM for solving objective (1) converts the objective to the dual domain. As each dual variable corresponds to a sample, sampling a subset of samples to update the model becomes easier and more natural than when done in the primal domain. The algorithm for performing ADMM on the dual of objective (1) turns out to be equivalent to the recently proposed SDCA-ADMM [18], which was originally proposed for solving objectives with complex regularizations (e.g., group lasso [11], graph guided SVM [14]) by combining ADMM and stochastic dual coordinate ascent (SDCA) [15], [16]. However, these works do not consider generalizing the method for use with distributed data, and in fact, a straightforward distributed implementation would suffer from frequent communication between machines, poor memory usage, and heavy computational demand. We show how to overcome these problems, achieving an efficient implementation for distributed data.

To summarize, our contributions are 1) proposing a simple, easy to implement, yet effective way to accelerate ADMM for distributed data with a theoretical guarantee; 2) proposing running ADMM on the dual of the objective and showing the advantages of doing so; and 3) describing an efficient

978-1-5090-5206-6/16 $31.00 © 2016 IEEE
DOI 10.1109/DSAA.2016.41

implementation of ADMM on the dual and showing the effectiveness of both methods on several binary classification datasets. The code to reproduce the experiments is available on *https://github.com/j123456/dis_sdcaadmm* through an anonymous id.

II. Background

We begin by describing how to apply ADMM in a distributed environment [5], [24]. The augmented Lagrangian of objective (1) is

$$L_\rho(w,c,\lambda) = \sum_{j=1}^{g} \sum_{i\in N_m} C\ max(1-l_i w_m^\top z_i,0)^2 + \frac{1}{2}\|c\|_2^2 + \sum_{j=1}^{g}\Big(\frac{\rho}{2}\|w_m - c\|_2^2 + \lambda_m^\top(w_m - c)\Big). \tag{2}$$

The algorithm in ADMM consists of

$$w^{(t+1)} = \text{argmin}_w L_\rho(w, c^{(t)}, \lambda^{(t)}), \tag{3}$$

$$c^{(t+1)} = \text{argmin}_c L_\rho(w^{(t+1)}, c, \lambda^{(t)}), \tag{4}$$

$$\lambda_m^{(t+1)} = \lambda_m^{(t)} + \rho(w_m^{(t+1)} - c^{(t+1)}), \tag{5}$$

where k is the iteration index, w is $\{w_1, \dots, w_g\}$, and λ is $\{\lambda_1, \dots, \lambda_g\}$. After solving (3), we get

$$\begin{aligned} w_m^{(t+1)} &= \text{argmin}_w C \sum_{i\in N_m} max(1 - l_i w^\top z_i, 0)^2 \\ &+ \frac{\rho}{2}\|w - c^{(t)}\|_2^2 + \lambda_m^\top(w - c^{(t)}), \end{aligned} \tag{6}$$

for each machine m. And (4) has a closed form,

$$c^{(t+1)} = \frac{\rho\Sigma_{m=1}^{g} w_m^{(t+1)} + \Sigma_{m=1}^{g}\lambda_m^\top}{g\rho + 1}. \tag{7}$$

Objective (6) is similar to the objective of a standard SVM. To see this, we rewrite (6) as

$$w_m^{(t+1)} = \text{argmin}_w C \sum_{i\in N_m} max(1 - l_i w^\top z_i, 0)^2 + \frac{\rho}{2}\|w - v_m^{(t)}\|_2^2, \tag{8}$$

where vector $v_m^{(t)} = c^{(t)} - u_m$ with $u_m^{(t)} = \lambda_m^{(t)}/\rho$.

[24] modified the method of [10] to solve (8), which is a dual coordinate descent method. The dual form of objective (8) can be shown as

$$\underset{\alpha}{\text{minimize}} \frac{1}{2\rho}\alpha^\top \overline{Q}\alpha - b^\top\alpha \quad s.t. \alpha_i \leq 0, \forall i, \tag{9}$$

where $b = [1 - l_1(v_m^t)^\top z_1, \dots, 1 - l_{N_m}(v_m^t)^\top z_{N_m}]$, $\overline{Q} = Q + D$, $Q_{ij} = l_i l_j z_j^\top z_j$, and D is a diagonal matrix $D_{ii} = \rho/(2C)$.

We can connect the primal form and dual form by [1]

$$w_m^{(t)} = \sum_{i=1}^{N_m} \frac{1}{\rho} l_i \alpha_i^{(t)} z_i + v_m^{(t)}. \tag{10}$$

This is achieved as follows. First, we rewrite (8) as

$$\begin{aligned} w_m^{(t+1)} &= \text{argmin}_w C \sum_{i\in N_m} \epsilon_i^2 + \frac{\rho}{2}\|w - v_m^{(t)}\|_2^2 \\ &\text{s.t } l_i(w_m^{(t+1)})\top z_i \geq 1 - \epsilon_i \\ &\qquad\qquad \epsilon_i \geq 0, \forall i. \end{aligned} \tag{11}$$

The Lagrangian of the above is

$$L_\rho = \frac{\rho}{2}\|w - v_m^{(t)}\|^2 + C\sum_i \epsilon_i^2 - \sum_i \alpha_i(l_i w^\top z_i - 1 + \epsilon_i). \tag{12}$$

Since the dual form is $\max_\alpha \min_w L_\rho$, by setting the partial derivative of L_ρ with respect to w to 0, we have

$$\rho(w - v_m^{(t)}) = \sum_i \alpha_i(l_i z_i). \tag{13}$$

After rearranging the above, we obtain

$$w_m^{(t)} = \sum_{i=1}^{N_m} \frac{1}{\rho} l_i \alpha_i^{(t)} z_i + v_m^{(t)}, \tag{14}$$

which is (10).

The dual problem is optimized by updating one variable α_i at a time, which has a closed-form,

$$\alpha_i^{(t+1)} = max(\alpha_i^{(t)} - G_i^{(t)}/\overline{Q}_{ii}, 0), \tag{15}$$

where $G_i^{(t)}$ is the gradient of the dual objective over α_i as shown below.

$$G_i^{(t)} = \sum_{n=1}^{N_m} \alpha_n^{(t)} \overline{Q}_{in} - b_i. \tag{16}$$

The computation of $G_i^{(t)}$ requires the summation of all the data, which is $\mathbb{O}(N_m)$. We can rewrite $G_i^{(t)}$ by using (10) to reduce the computation complexity $\mathbb{O}(s)$, where s is the average number of non-zero features in the data. That is, we can express $G_i^{(t)}$ as

$$G_i^{(t)} = \rho l_i {w_m^{(t)}}^\top z_i - 1 + D_{ii}\alpha_i^{(t)}, \tag{17}$$

where ${w_m^{(t)}}^\top z_i$ is $\mathbb{O}(s)$. The above trick is useful when the feature is sparse.

After updating a_i, we can update $w_m^{(t+1)}$ as

$$w_m^{(t+1)} = w_m^{(t)} + \frac{1}{\rho}(\alpha_i^{(t+1)} - \alpha_i^{(t)}) l_i z_i. \tag{18}$$

III. A sampling approach for fast ADMM on primal objective (1)

From the previous section, we know that ADMM requires solving (8) at every iteration of ADMM. Each iteration can take many inner iterations to complete. When local data becomes large, it would take a substantial amount of time to train a model. To deal with this drawback, we propose a method that reduces the training cost. At the earlier iterations, each local machine only uses a subset of its data instead of all the data to update its learned model. As the algorithm continues, each machine gradually increases the amount of data used and finally reaches the full capacity. This method enjoys similar fast decrease of objective value as SGD does at the first few iterations and shares the same convergence rate

[1] In the paper of Zhang et al. [24], it did not include the $1/\rho$ term, it may be a typo, as shown in the appendix.

in the long term as using the full dataset every iteration. Thus, for each iteration, each machine solves (19) instead of (8)

$$\begin{aligned} w_m^{(t+1)} = \text{argmin}_w C &\sum_{i\in N_m^k} max(1 - l_i w^\top z_i, 0)^2 + \\ &\frac{\rho}{2}\|w - v_m^{(t)}\|_2^2, \forall m, \end{aligned} \tag{19}$$

where we have replaced N_m with $N_m^{(t)}$, which represents the amount of data used on machine m at the k^{th} iteration.

The amount of data used at the $k+1$ iteration, $N_m^{(t+1)}$, satisfies

$$|N_m^{(t+1)}| = max\{|N_m^{(t)}| \times \beta, |N_m|\}, \tag{20}$$

where $\beta > 1$ is the increase rate. Note that N_m stands for total number of samples on machine m. For example, we can initialize N_m^1 to $0.5|N_m|$, and set β to 1.1, meaning the amount of data used for training at each iteration in ADMM increases by 10 percents each iteration.

The optimization procedure of the modified algorithm is not very different from the traditional one; it iterates over (3)-(5), except that the sub-problem (3) or (8) is replaced by (19), which can be solved in a similar manner as described in the previous section.

Now we analyze the convergence rate in terms of iterations of the proposed method. For brevity, we rewrite the objective to a general form as

$$\begin{aligned} &\underset{w,c}{\text{minimize}}\ f(w) + g(c) \\ &\text{s.t. } Aw + Bc = k. \end{aligned} \tag{21}$$

Then, the augmented Lagrangian is

$$\begin{aligned} L_\rho(w, c, \lambda) = f(w) + g(c) + \lambda^\top(Aw + Bc - k) \\ + \frac{\rho}{2}\|Aw + Bc - k\|_2^2. \end{aligned} \tag{22}$$

For the distributed data setting, $f(w) = \sum_{m=1}^{g} \sum_{i\in N_m} C\ max(1 - l_i w^\top z_i, 0)^2$ and $g(c) = \frac{1}{2}\|c\|_2^2$ in (22). Since our algorithm uses a subset of data to update w for each iteration, the partial optimality equation of $L_\rho(w, c, \lambda)$ with respect to w would not be zero if measured by using full data. We assume there is an error term $e^{(t)}$. We have the following theorem

Theorem 1 Let $\overline{w}_T = \frac{1}{T}\Sigma_{t=1}^T w^{(t)}$, $\bar{c}_T = \frac{1}{T}\Sigma_{t=1}^T c^{(t)}$. For any x^*, c^* satisfying $Ax^* + Bc^* = k$, we have

$$\begin{aligned} &f(\overline{w}_T) + g(\bar{c}_T) - f(w^*) - g(c^*) \\ &\leq \frac{\rho}{2T}(\|Bc^* - Bc^{(0)}\|_2^2 - \|Bc^* - Bc^{(T)}\|_2^2) + \frac{1}{2\rho T}(\|\lambda^{(0)}\|_2^2 - \\ &\|\lambda^{(T)}\|_2^2) + \frac{1}{T}\Sigma_{t=0}^{T-1}\|e^{(t)}\|\|w^{(t+1)} - w^*\| \end{aligned} \tag{23}$$

If $\Sigma_{t=0}^{T-1}\|e^{(t)}\|\|w^{(t+1)} - w^*\|$ is sublinear to T, the above implies $\mathcal{O}(\frac{1}{T})$ convergence rate just like the standard ADMM [9]. **Q.E.D**

In the algorithm, after the first few iterations, each machine begins to use all of its data to update its model. It is reasonable to assume that $\|e^{(t)}\|$ go towards zero as the algorithm begins using all data. Moreover, $\|w^{(t+1)} - w^*\|$ is also sublinear with respect to T due to the fact that the objective is strongly convex. Thus, $\Sigma_{k=0}^{T-1}\|e^{(t)}\|\|w^{(t+1)} - w^*\|$ is bounded and with high probability it could be sublinear with respect to T.

IV. A SAMPLING APPROACH FOR FAST ADMM ON DUAL OBJECTIVE OF (1)

We propose another method that still follows the similar idea of sampling a subset of data on each round of communication. This section begins by converting the primal form (1) to dual form. In order to make the dual form compact, we relax the constraint and approximate it. As each dual variable corresponds to a sample, sampling a subset to update the model becomes easier and natural. We then show how to integrate the sampling idea in performing ADMM on the dual of (1). We will show that solving the dual of (1) by ADMM turns out to be equivalent to SDCA-ADMM. We then propose techniques to efficiently perform SDCA-ADMM for distributed data.

A. Converting primal (1) to approximated dual form

To achieve the goal, we transform each feature vector from $\mathbb{R}^p$ to $\mathbb{R}^{pg}$. That is, the dimensions of augmented feature space is $p \times g$, which is g times larger than the original feature dimension p. Let us denote the original feature vector of the i_{th} sample on the m machine as $\tilde{z}_{i,m} \in \mathbb{R}^p$, the transformation $\mathbb{R}^p \to \mathbb{R}^{pg}$ is:

$$\begin{aligned} z_{i,m}((m-1)p + 1 : mp) =& \tilde{z}_{i,m}, \\ &\text{if } i \text{ sample on the } m \text{ machine}, \end{aligned} \tag{24}$$

the other entries in $z_{i,m}$ are set to zeros. Thus, the data matrix $Z \in \mathbb{R}^{pg\times N}$ becomes

$$Z = [Z_1, Z_2, \ldots, Z_m, \ldots, Z_g], \tag{25}$$

where $Z_m \in \mathbb{R}^{pg\times n_m}$ is a data matrix on machine m that consists of augmented feature $z_{i,m}$ on each column, and there are n_m samples on machine m. Moreover, we have $\Sigma_{m=1}^g n_m = N$, which is the number of all the data samples. By the representation of (24) and (25), Z is a block diagonal matrix. Let us denote the diagonal blocks as $Z_{(m)}, m \in \{1, \ldots, g\}$. Note that $Z_{(m)}$ is the submatrix of Z_m that consists of original features $\tilde{z}_{i,m}$.

We now turn to specify a matrix $B \in R^{pg\times g}$ that encodes the constraint in objective (1), which is $w_{(1)} = w_{(2)} = \cdots = w_{(g)}$. Since the feature dimensions becomes $p \times g$ dimensions, so does the corresponding classifier w. Denote the subvector $w_{(m)} \in \mathbb{R}^p$ the m_{th} block of $w \in \mathbb{R}^{pg}$. Due to the augmented features we design, we can view $w_{(m)}$ as a model learned by the m_{th} local machine. We then specify the transform B to be

$$B \in \mathbb{R}^{pg\times g} = \begin{pmatrix} 1_p & & & -1_p \\ -1_p & 1_p & & \\ & -1_p & 1_p & \\ \cdots & \cdots & \cdots & \cdots \\ & & -1_p & 1_p \end{pmatrix} \tag{26}$$

where 1_p means the p dimensional vector of all $1's$. Thus, $B^\top w$ is equal to the constraint $w_{(1)} = w_{(2)} = \cdots = w_{(g)}$ that encourages the model associated with each machine to agree with each other. To make the dual form more compact and simplified, we relax the constraint and propose to use a regularization term Ψ so that $\Psi(B^\top w)$ can have the similar effect of the constraint. We choose $\Psi(\cdot)$ to be a squared of l_2 norm, which is $\frac{1}{2}C'\|\cdot\|_2^2$ with C' controls the regularization effect. Thus, $\Psi(B^\top w)$ would be

$$\frac{1}{2}C'\{\sum_{m=1}^{g}\|(w_{(m)}-w_{(m+1)})\|_2^2\}, \tag{27}$$

where we use the notation that $w_{(g+1)} \equiv w_{(1)}$. The regularization penalizes the difference of a machine with its neighbors (in terms of the index m) and encourages the subvectors $w_{(m)}$ of w to agree with each other. Note that the specification of the transform B is flexible. If we have a prior about the relation between the models, we can easily encode it in the transform. For example, if we believe that data behave slightly heterogeneous (i.e. dataset bias [19]) on some machines, we can allow a difference on the models learned by those machines. As ADMM enjoys the benefits of solving structured regularization, it then becomes its advantage as compared to distributed SDCA [22], because SDCA has closed form update only when the objective function and regularization is simple enough (e.g. some common loss with l_1 or l_2 norm) [15], [16]. It may be hard for SDCA to encode structure regularization without sacrificing the benefit of closed form updates.

To summarize, the conversion to the approximated dual form is

$$\begin{aligned}\underset{w\in\mathbb{R}^{pg}}{\text{minimize}}\ &\frac{1}{N}\sum_{m=1}^{g}\sum_{i=1}^{n_m}f_{i,m}(z_{i,m}^\top w)+\psi(B^\top w)=\\ &-\underset{x\in\mathbb{R}^N,y\in\mathbb{R}^g}{\text{minimize}}\sum_{m=1}^{g}\sum_{i=1}^{n_m}f_{i,m}^*(z_{i,m}^\top w)+\psi^*(\frac{y}{n})\\ &\text{s.t. } Zx+By=0,\end{aligned} \tag{28}$$

where $N=\Sigma_m n_m$ and the definitions of $z_{i,m}$, Z, B, $\psi(B^\top w)$ are in (24), (25), (26), and (27) respectively. Here $f_{i,m}=f$ is the loss function associated to the sample i on machine m, and the symbol $*$ is used to represent the corresponding conjugate function.

B. SDCA-ADMM

We would like to point out that objective (28) can be solved by SDCA-ADMM [18]. SDCA-ADMM [18] was originally proposed to solve the regularized risk minimization.

$$\underset{w\in\mathbb{R}^p}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}f_i(z_i^\top w)+\tilde{\psi}(w), \tag{29}$$

where f_i is the loss associated with the i_{th} sample, and $\tilde{\psi}$ is a complex regularization the work intended to deal with. (There are also some related works proposing stochastic/online ADMM for complex regularization, see [2], [14], [17], [20], [26].) The proximal operation for $\tilde{\psi}$ may not be easy to be derived, yet in many cases, $\tilde{\psi}$ can be rewritten as $\tilde{\psi}(w)=\psi(B^\top w)$ where $B\in\mathbb{R}^{p\times g}$ is a linear transform and the proximal operation for ψ would be easy to be computed now.

The work [18] then transforms the objective (29) into dual form.

$$\begin{aligned}&\underset{w\in\mathbb{R}^p}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}f_i(z_i^\top w)+\psi(B^\top w)\\ &=-\underset{x\in\mathbb{R}^n,y\in\mathbb{R}^g}{\text{minimize}}\sum_{i=1}^{n}f_i^*(z_i^\top w)+\psi^*(\frac{y}{n})\ \text{s.t. } Zx+By=0,\end{aligned} \tag{30}$$

where f_i^* and ψ^* are the conjugates of f_i and ψ respectively, and $Z=[z_1,z_2,\ldots,z_n]\in\mathbb{R}^{p\times n}$ is a data matrix with each column representing a sample. The optimal solution w^*, x^*, and y^* satisfy the following condition.

$$\begin{aligned}&z_i^\top w^*\in\nabla f_i^*(x_i^*),\ \frac{1}{n}y^*\in\nabla\psi(u)|_{u=B^\top w^*},\\ &Zx^*+By^*=0.\end{aligned} \tag{31}$$

Note that the dual problem (30) is a composite optimization with a linear constraint $Zx+By=0$.

Suppose the data are split into B batches: $(I_1,I_2,\ldots,I_B)$ and I_b is the index of a subset of $\{1,\ldots,n\}$ that represents the b_{th} batch of samples. At each iteration of SDCA-ADMM, one mini-batch, denoted it as $I=I_b$, is chosen uniformly at random. The update rules at the t_{th} iteration of the optimization are

$$\begin{aligned}y^{(t)}=&\ \underset{y}{\text{argmin}}\ n\Psi^*(\frac{y}{n})-\langle w^{(t-1)},Zx^{(t-1)}+By\rangle\\ &+\frac{\rho}{2}\|Zx^{(t-1)}+By\|^2+\frac{1}{2}\|y-y^{(t-1)}\|_Q^2\\ x_I^{(t)}=&\ \underset{x_I}{\text{argmin}}\sum_{i\in I}f_i^*(x_i)-\langle w^{(t-1)},Z_Ix_I+By^{(t)}\rangle\\ &+\frac{\rho}{2}\|Z_Ix_I+Z_{\backslash I}x_{\backslash I}^{(t-1)}+By^{(t)}\|^2+\frac{1}{2}\|x_I-x_I^{(t-1)}\|_{G_I}^2\\ w^{(t)}=&\ w^{(t-1)}-\gamma\rho\{n(Zx^{(t)}+By^{(t)})-(n-n/K)(Zx^{(t-1)}\\ &+By^{(t-1)})\},\end{aligned} \tag{32}$$

where $x_I\in\mathbb{R}^{|I|}$ is the dual variable corresponds to the samples batch I, $y^{(t)}$ is the dual variable associated with the regularization term and $w^{(t)}$ is the primal variable at the t iteration, and γ,ρ are the parameters of the algorithm. We have used the notation that $\|x\|_\Omega\doteq\sqrt{x^\top\Omega X}$, where Ω is a positive semidefinite matrix. Simply neglecting the last term in the updates for $y^{(t)}$ and $x_I^{(t)}$, we can view the updates as performing standard ADMM on the dual space, where the dual variables of $y^{(t)}$ and $x_I^{(t)}$ are now the primal variable $w^{(t)}$.

The positive semidefinite matrices Q and G_I in the weighted norms need to be specified. Suitable choices of the matrices can simplify the updates, making the updates have closed form. Suzuki [18] set Q as $Q=\rho(\eta_BI_g-B^\top B)$, where η_B is chosen to satisfy

$$\eta_BI_g\succeq B^\top B \tag{33}$$

And G_I is set to be $G_I=\rho(\eta_{Z,I}I_{|I|}-Z_I^\top Z_I)$, where $\eta_{Z,I}$ is chosen so that

$$\eta_{Z,I}I_{|I|}\succeq Z_I^\top Z_I. \tag{34}$$

As a result, the update of $y^{(t)}$ can be simplified as

$$y^{(t)}=prox(q^{(t)}|\frac{n\psi^*(\cdot/n)}{\rho\eta_B}), \tag{35}$$

where

$$q^{(t)}=y^{(t-1)}+\frac{B^\top}{\rho\eta_B}[w^{(t-1)}-\rho(Zx^{(t-1)}+By^{(t-1)})], \tag{36}$$

and the $prox$ means the proximal operation

$$prox(\theta|\Psi)\doteq\underset{u}{\text{argmin}}\frac{1}{2}\|\theta-u\|^2+\Psi(u). \tag{37}$$

The update (38) can also be written as

$$y^{(t)} = q^{(t)} - prox(q^{(t)}|n\psi^*(\rho\eta_B\ \cdot)/(\rho\eta_B)) \tag{38}$$

By specifying G_I as in (34), updating $x_I^{(t)}$ can be simplified to

$$x_I^{(t)} = prox(r_I^{(t)}|\{\sum_{i\in I} f_i^*/(\rho\eta_{Z,I})\}), \tag{39}$$

where

$$r_I^{(t)} = x_I^{(t-1)} + \frac{Z_I^\top}{\rho\eta_{Z,I}}[w^{(t-1)} - \rho(Zx^{(t-1)} + By^{(t)})]. \tag{40}$$

The update rule (39) can be decomposed into several proximal operation with respect to single variable x_i, $i \in I$. That is, we can compute each element of $x_I^{(t)}$ by

$$x_i^{(t)} \leftarrow prox(r_i^{(t)}|\frac{f_i^*}{\rho\eta_{Z,I}}), \text{ for each } i \in I \tag{41}$$

where $r_i^{(t)}$ is the i_{th} element of $r_I^{(t)}$. Thus, updating $x_I^{(t)}$ can be done by parallelly computing each x_i.

The techniques of choosing suitable positive semidefinite matrix in the weighted norms is called linearization of ADMM [14], [17]. By linearization, we can simplify the update rules and the updates may allow to be written in closed form.

C. *Efficient implementation of distributed SDCA-ADMM*

The naive distributed implementation of SDCA-ADMM for solving the approximated dual form of objective (1), which is (28), would operate on the high dimensional feature space $\mathbb{R}^{pg}$ with frequent communication between machines and large memory consumption because of feature augmenting. Yet, by exploiting the block-diagonal structure of data matrix Z, (24) and (25), the algorithm can be conducted on the original feature space $\mathbb{R}^p$ while unnecessary communication and computation can be avoided.

Let us take the update rule for variables x as an example, described in (39) and (40). At iteration t, each machine m uniformly and randomly chooses a mini-batch, say I_m, and updates the new values of dual variables associated with I_m. Thus, the dual variables that are going to be updated at the iteration is fully indexed by $I = \{I_1, \ldots, I_m, \ldots, I_g\}$, with $|I| = \sum_{m=1}^{g} |I_m|$. Now, consider the computations $\frac{Z_I^\top}{\rho\eta_{Z,I}}[w^{(t-1)} - \rho(Zx^{(t-1)} + By^{(t)})]$ in (40). Let us focus on $Z_I^\top w^{(t-1)}$ first. At first glance, the multiplication of $Z_I \in \mathbb{R}^{pg\times|I|}$ and $w^{(t-1)} \in \mathbb{R}^{pg}$ suffers from the high dimensions of pg. Fortunately, if we exploit the block diagonal structure of Z, the multiplication can be decomposed into several smaller matrix-vector multiplications: $Z_{(m),I_m}^\top w_{(m)}^{(t-1)}$ on each machine m, where $Z_{(m),I_m} \in \mathbb{R}^{p\times|I_m|}$ represents the columns of the submatrix $Z_{(m)}$ indexed by the mini-batch I_m on machine m. Consequently, each machine m only need to compute its $Z_{(m),I_m}^\top w_{(m)}^{(t-1)}$ on its local data. Moreover, it does not require the multiplication results associated with other machines since the update of subvector of $r_I^{(t)}$ that corresponds to the I_m batch only depends on $Z_{(m),I_m}^\top w_{(m)}^{(t-1)}$. This suggests that we do not really need to transform the features into the augmented high dimensional feature space and, as a consequence, it avoids the unnecessary communication, computation, and memory consumption. Figure 1 depicts the decomposition.

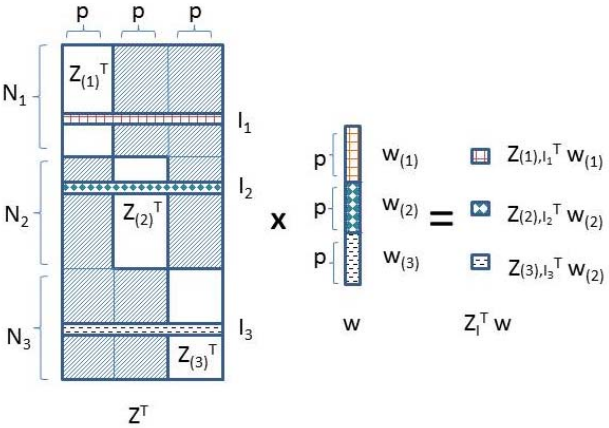


Fig. 1. Illustration on how to compute $Z_I^o pw$. This is $g = 3$ cases (i.e. data are distributed on three machines). Suppose I_1, I_2, and I_3 batches are chosen at current iteration, so $I = \{I_1, I_2, I_3\}$. Since matrix Z is a block diagonal matrix and all the off-diagonal blocks are zeros (which are filled with the slash lines on the graph), the computations can be decomposed into smaller components $Z^o_{(1),I_1}pw_{(1)}$, $Z^o_{(2),I_2}pw_{(2)}$, and $Z^o_{(3),I_3}pw_{(3)}$, each is independently computed on the respective local machine. Thus, the unnecessary computation and communication can be avoided.

Algorithm 1 Distributed SDCA-ADMM

```
Input: parameters $\rho, \gamma, \eta_{Z,I}, \eta_B$
Initialize $x_0 = 0$, $y_0 = 0$, $w_0 = 0$
for $t = 1$ to $T$ do
    1. The master receives $c_{(m)}^{(t-1)}$ and update $y^{(t)}$.
    $q^{(t)} = y^{(t-1)} + \sum c_{(m)}^{(t-1)}$
    $y^{(t)} = q^{(t)} - prox(q^{(t)}|n\Psi(\rho\eta_B\ \cdot\ )/(\rho\eta_B))$
    2. The master broadcasts $y^{(t)}$.
    3. Each local machine $m$ randomly chooses a mini-batch $I_m$, so totally $\sum_{m=1}^{g} |I_m|$ of the dual variables $x$ are updated at the $k$ iteration.
    4. Each local machine updates $x_{i,m}, i \in I_m$.
    $x_{i,m}^{(t)} \leftarrow prox(r_{i,m}^{(t)}|\frac{f_{i,m}^*}{\rho\eta_{Z,I}})$ for each $i \in I_m$, where $r_{i,m}^{(t)}$ is the $i_{th}$ element of $r_{I_m}^{(t)}$, and $r_{I_m}^{(t)} = x_{I_m}^{(t-1)} + (Z_{(m),I_m}^\top/(\rho\eta_{Z,I}))\times\{w_{(m)}^{(t-1)} - \rho(Z_{(m)}x_{(m)}^{(t-1)} + B_{(m)}y^{(t)})\}$
    5. Each local machine updates $w_{(m)}^{(t)}$
    $w_{(m)}^{(t)} = w_{(m)}^{(t-1)} - \gamma\rho\{n(Z_{(m)}x_{(m)}^{(t)} + B_{(m)}y^{(t)}) - (n - n/K)(Z_{(m)}x_{(m)}^{(t-1)} + B_{(m)}y^{(t-1)})\}$
    and then computes $c_{(m)}^{(t)}$ and send it to the master.
    $c_{(m)}^{(t)} = \frac{B_{(m)}^\top}{\rho\eta_B}\{w_{(m)}^{(t)} - \rho(Z_{(m)}x_{(m)}^{(t)} + B_{(m)}y^{(t)})\}$
end for
Output: $\frac{1}{g}\sum_m w_{(m)}^{(T)}$.
```

Similarly, now let us consider $Zx^{(t-1)}$. It can be decomposed into $Z_{(m)}x_{(m)}^{(t-1)}$, where $x_{(m)}^{(t-1)}$ represents the dual variables associated with the data on machine m at the

$t-1$ iteration. Thus, each machine can simply maintains its $Z_{(m)}x^{(t-1)}_{(m)}$ independently and thus does not require those of other machines. However, maintaining $Z_{(m)}x^{(t-1)}_{(m)}$ seems to required $\mathcal{O}(n_m)$ complexity, which is burdensome when facing large amount of data. A remedy is to incrementally update $Z_{(m)}x^{(t)}_{(m)}$ as $Z_{(m)}x^{(t)}_{(m)} = Z_{(m)}x^{(t-1)}_{(m)} + \tilde{z}_{i,m}(x^{(t)}_{i,m} - x^{(t-1)}_{i,m})$ for each sample i in the mini-batch I_m. As a result, the update can be computed much quicker since the complexity is reduced from $\mathcal{O}(n_m)$ to $\mathcal{O}(|I_m|)$ and $|I_m| \ll n_m$.

Now let us turn to the last component $By^{(t)}$ in (40). Denote $B_{(m)}$ as the m_{th} block of p rows. For example, $B_{(1)}$ would be the first p rows of B, which corresponds to the constraint between $w_{(1)}$ and $w_{(m)}$. Because of the design of B in (26) (i.e. blocks of 1's), the value of the elements in $B_{(m)}y$ is identical. Thus, only one element in $B_{(m)}y$ needs to be computed.

For the update of primal w, we have

$$\begin{aligned} w^{(t)}_{(m)} = & w^{(t-1)}_{(m)} - \gamma\rho\{n(Z_{(m)}x^{(t)}_{(m)} + B_{(m)}y^{(t)}) - \\ & (n - n/K)(Z_{(m)}x^{(t-1)}_{(m)} + B_{(m)}y^{(t-1)})\}. \end{aligned} \tag{42}$$

The overall implementation is shown in Algorithm 1, which iterates between updating dual variables y, x, and primal variables w. The variables are initialized as zero vectors. For updating y, which are shown in (35) and (36), the component $\frac{B^\top}{\rho\eta_B}[w^{(t-1)} - \rho(Zx^{(t-1)} + By^{(t-1)})]$ can be decomposed into the summation of smaller components $c^{(t)}_{(m)} = \frac{B^\top_{(m)}}{\rho\eta_B}\{w^{(t)}_{(m)} - \rho(Z_{(m)}x^{(t)}_{(m)} + B_{(m)}y^{(t)})$. Thus, each machine locally computes $c^{(t)}_{(m)}$ and the master aggregates the results to update y. Each machine then chooses a batch of dual variables X_{I_m} (for each m) to update. After that, it updates $w^{(t)}_{(m)}$ and finally computes $c^{(t)}_{(m)}$ and send it to the master. We use the average $w^{(T)}_{(m)}$ as the final solution.

D. Theoretical Analysis

SDCA-ADMM can be shown to converges R-linearly [18]. The convergence rate depends on spectral norm of data. If the correlation between samples are small, the number of iterations can be reduced so that it can converge faster as the size of batch increases until some point (despite the serial running time increases with the batch size). Otherwise, the convergence rate may not be improved by increasing the batch size. This nature also appears in mini-batch SDCA, as shown by the studies of [13], [15], and [22].

V. Related Works

There are growing interests in distributed data setting. Zinkevich et al. [23] and Zhang et al. [27] consider running the optimization independently for each machine and combining the learned models at the final stage. Duchi et al. [7] and Xio [21] propose methods based on dual subgradient averaging [21], and provide sharp bounds on their convergence rates under several network topologies. Very recently, there are works that integrate the second order information for distributed optimization. Agarwal et al. [1] consider distributed implementation of L-BFGS [12]. Among all the distributed optimization methods, perhaps the most well-recognized is ADMM In addition to solve the distributed data scenario we consider, it can achieve variable splitting for more complex composite objectives. Because of the pupularity and the widely usage of ADMM, we tend to improve the method, which is the goal of this paper.

TABLE I. Dataset statistics

Data	number of samples	dimension
delta	500,000	500
gamma	500,000	500
mnist47	1,634,445	784
ocr	3,500,000	1,156
epsilon	500,000	2,000
rcv1	697,641	47,236

Perhaps the most closely related work with our first proposed method are [4] and [8]. They combine the advantages of stochastic gradient descent and gradient descent, though not in distributed setting. The advantage of SGD lies in its cheaper computational cost since only one sample is chosen at a time to compute the gradient for each iteration, while still able to achieve the fast decrease of objective value. However, due to the noisy gradient it computes, SGD is eventually dominated by gradient descent. Thus, the algorithm behaves like SGD at the beginning and move towards gradient descent eventually. Our work differs from theirs in that we consider the distributed data setting solved by ADMM. Each machine first uses a subset of data like stochastic variants of ADMM but eventually uses all data as standard ADMM does.

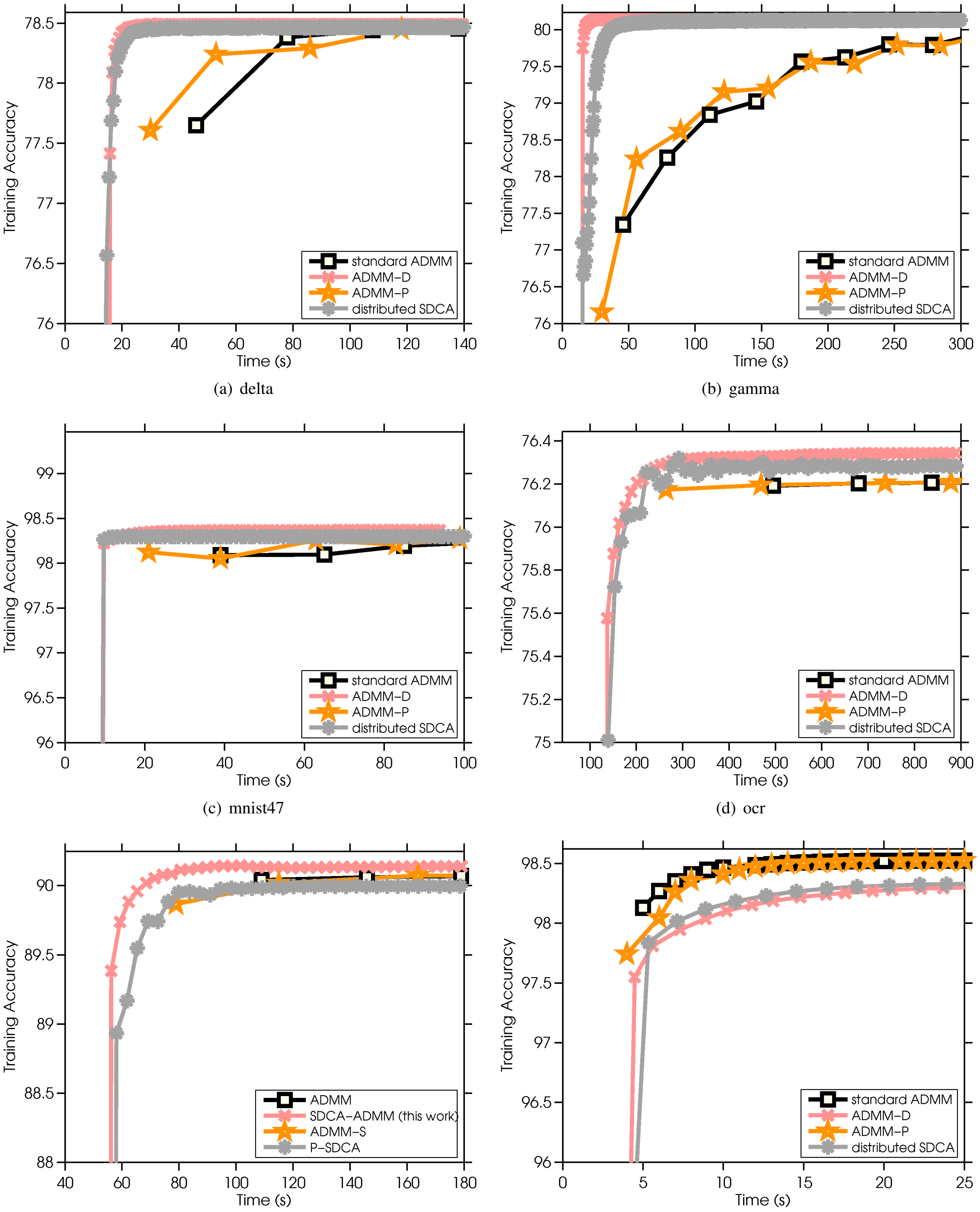
VI. Experiments

We compare our methods with standard ADMM [24] and distributed SDCA [22] on several datasets. We denote our first method which performs ADMM on primal objective (1) with sampling as ADMM-P, and we denote the second method which performs distributed SDCA-ADMM on dual objective (28) as ADMM-D. The datasets are for binary classification. [2] Table 1 shows their statistics. For delta, gamma, and ocr, because only the labels of original training data are available, we split each of them by 80% as training set and the remain 20% as testing set. For mnist, we choose classifying digit 4 against digit 7 as the recognition goal and also split the data into 80/20 split. For epsilon, we use the pre-defined training/testing split. For rcv1, the ratio of original training to testing data size is much smaller, so we re-split the data into 80/20 split. The data are further distributed on four machines in our workstation. The batch size $|I|$ is set to 100 (i.e each of four machines uses 25 samples at a time) for both SDCA and ADMM-D on all the datasets except rcv1 where the batch size $|I|$ is set to 1000.

In our distributed SDCA-ADMM, we use the smoothed hinge loss

$$f_i(z_i^\top w) = \begin{cases} 0 & (y_i z_i^\top w \geq 1) \\ \frac{1}{2} - y_i z_i^\top w & (y_i z_i^\top w < 0) \\ \frac{1}{2}(1 - y_i z_i^\top w)^2 & (\text{otherwise}) \end{cases} \tag{43}$$

[2] delta, gamma, and ocr are available on *http://largescale.ml.tu-berlin.de/about*, while mnist, epsilon, and rcv1 are available on the LIBSVM webpage.



(a) delta

(b) gamma

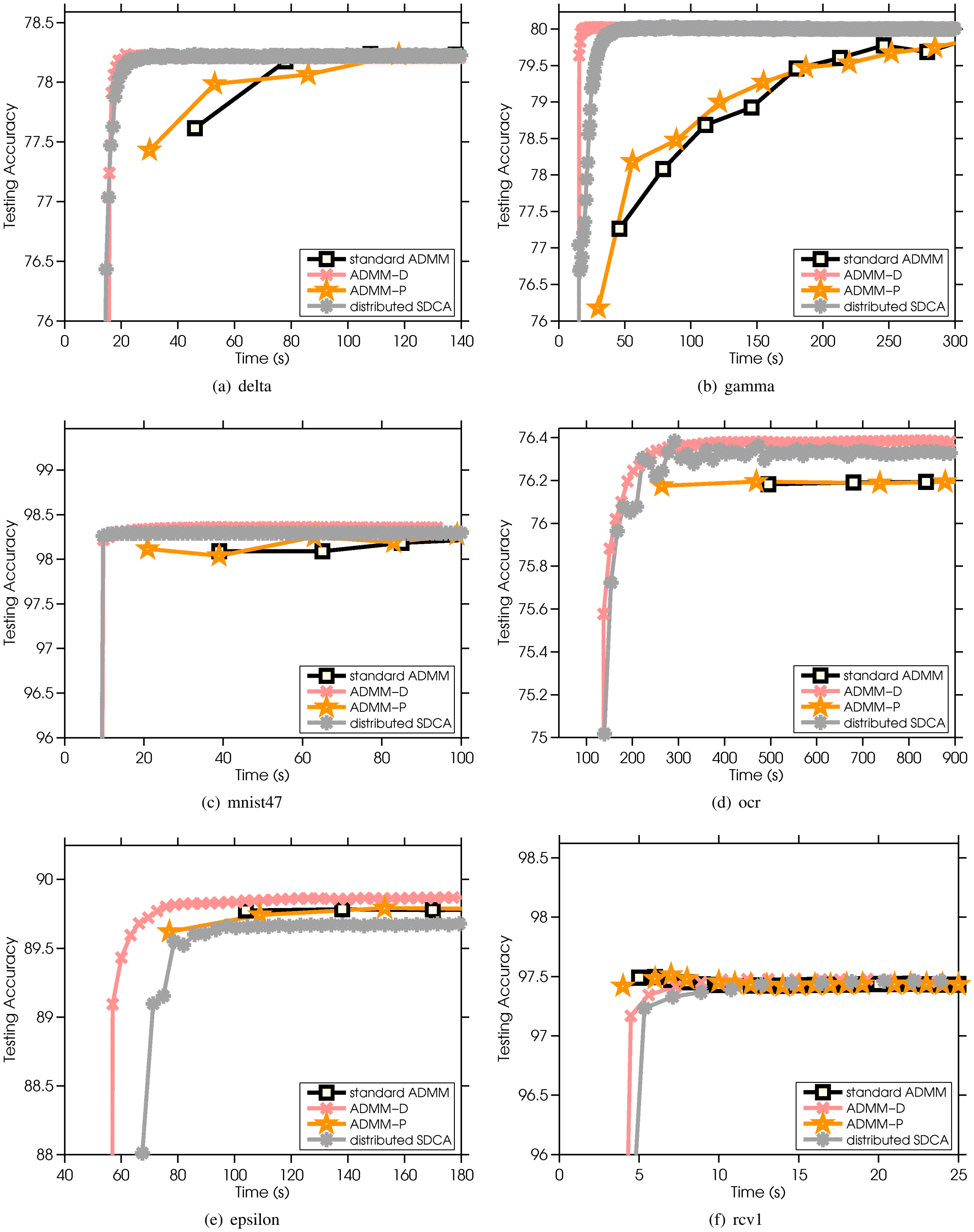
(c) mnist47

(d) ocr

(e) epsilon

(f) rcv1

Fig. 2. Training accuracy vs. time for each method on different datasets.



(a) delta

(b) gamma

(c) mnist47

(d) ocr

(e) epsilon

(f) rcv1

Fig. 3. Testing accuracy vs. time for each method on different datasets.

The proximal operation with respect to its dual form has a closed form solution, which is involved in the update of dual variables x (see (41)),

$$prox(u|f_i^*/\theta) = \begin{cases} \frac{\theta u - y_i}{1+\theta} & (-1 \leq \frac{\theta u y_i - 1}{1+\theta} \leq 0) \\ -y_i & (-1 > \frac{\theta u y_i - 1}{1+\theta}) \\ 0 & \text{(otherwise)} \end{cases} \quad (44)$$

As mentioned previously, we let regularization $\Psi(\cdot)$ as $\frac{1}{2}C\|\cdot\|_2^2$. Thus, the update of dual variables y (38) would be

$$y^{(t)} = \frac{q^{(t)}}{1 + \frac{1}{Cn\rho\eta_b}}. \quad (45)$$

There are some parameters needed to be specified in ADMM-D: ρ, γ, $\eta_{Z,I}$, and η_B, where $\eta_{Z,I}$ and η_B should be chosen so that (33) and (34) are satisfied. For γ, we set it as $\gamma = 1/n$. For η_B, due to the choice of transform B, simple calculation would show that it should be at least gp (i.e. the value no less than the multiplication of number of machines g and feature dimensions p). For $\eta_{Z,I}$, it should be larger than the largest eigenvalue of $Z_I^\top Z_I$ associated with the mini-batch I. Note that it is time consuming to compute it for each mini-batch at the augmented feature space. Instead, before running the optimization, we simply sample a mini-batch, say I, and calculate the largest eigenvalue of $Z_I^\top Z_I$ at the original feature space. Denote the computed value as η_{tmp}. We set every $\eta_{Z,I}$ to the same value: $\eta_{Z,I} = \theta$, where $\theta = 5 \times 10^d$ with the smallest power d such that $\theta > \eta_{tmp}$. For ρ, it is set to $10^{d'}$ with d' chosen to satisfy $\rho \times \eta_{Z,I} = 5$. We found the above heuristic work well in practice. The parameters for standard distributed ADMM is set to the default setting as in [24], while for the ADMM-P, we set the additional parameter k to 1.5, which means that samples used is increased by 1.5 times at subsequent iterations

Since the objective of each method is not the same (i.e. for ADMM-D, the objective is shown on (28); for standard ADMM and ADMM-P, the objective is shown on (1); for parallel-SDCA, the objective is hinge loss with l_2 regularization without any constraint as compared to the others.), we tune the regularization parameter C such that each method can achieve to the best accuracy on each dataset. We report the results on Figure 2 and 3. Figure 2 shows the training accuracy vs. training time, while Figure 3 shows the testing accuracy vs. training time. Each curve represents a different method. For ADMM or ADMM-P, each point on a curve is the classification result of a model at a corresponding iteration, while for distributed SDCA or ADMM-D, each point is a multiple of iterations that is roughly equivalent to a full pass of data. We run the distributed SDCA and ADMM-D ten times for each dataset so the results are the averaged ones.

From the figures, we see that ADMM-D converges the fastest on most of the datasets except rcv1 among the methods. The figure also shows that ADMM-D and distributed SDCA outperform standard ADMM and ADMM-P on most of the cases, while ADMM-D is better than distributed SDCA on gamma and epsilon. On other data, ADMM-D is comparable or slightly better than distributed SDCA. We note that though the improvements are not large on some datasets, as our algorithms belong to primal-dual optimization that update on both primal and dual variables in each iteration, it is easy to compute duality gap to measure the progress during iterations, while SDCA may not be easy to achieve that. This indicates that ADMM-D (distributed SDCA-ADMM) can enjoy the merits of SDCA and ADMM and outperform them as a consequence. The figure also shows ADMM-P is at least comparable to standard ADMM. ADMM-P outperforms ADMM on delta, gamma and mnist47. On rcv1, standard ADMM seems to be better than ADMM-D and distributed SDCA. Note that the feature dimensions of rcv1 data is much larger than the other datasets. As the dimensions increases, computations such as matrix-vector product also scales with the dimensions. For standard ADMM [24], the algorithm follows a strategy used in LIBSVM [6] to update $w_{(m)}$, where it maintains and updates a active set [10] such that the dual variables associated with samples outside the active set do not need to be updated anymore. This means that the computational time is decreased through iterations of standard ADMM, which alleviates the suffer of high dimension. While in SDCA or ADMM-D, at each iteration, it still samples a pre-determined batch size of data. Thus, an interesting future work would be to exploit the active set strategy into distributed SDCA and ADMM-D. To summarize, ADMM-D is an effective method on medium feature size data. If frequent communication is admissible (as in our setup), we suggest to use ADMM-D, otherwise use ADMM-P.

VII. Conclusion

We propose sampling-based ADMM approaches for learning from distributed data. We integrate the idea from stochastic gradient descent into ADMM. Our first method uses subset of data on early rounds of communication, which can reduce the cost on early stage while enjoy the similar convergence rate as the standard one. We also transform the primal objective into the approximated dual form and propose a distributed variant of the recently proposed SDCA-ADMM to solve it. The approximated dual form can be viewed as performing group regularization on the augmented feature space. As described, the algorithm does not necessarily have to be conducted on the augmented feature space. Due to the diagonal structure of expanded data matrix and the transform we specified, the matrix-vector or matrix-matrix product that involved in the updates can be decomposed into smaller components, which consequently allows the operations computed on original feature space and avoids the unnecessary computations and memory consumption. Future works include the variant of the proposed methods that deal with asynchronous issue like [25].

VIII. Acknowledgement

This material was based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-15-1-4013 and Microsoft Research Asia (MSRA) under award number FY16-RES-THEME-013.

References

[1] Alekh Agarwal, Olivier Chapelle, Miroslav Dudik, John Langford. “A Reliable Effective Terascale Linear Learning System“. Journal of Machine Learning Research, 15:1111–1133, 2014.

[2] Samaneh Azadi and Suvrit Sra. “Towards an optimal stochastic alternating direction method of multipliers.” ICML, 2014.

[3] Lèon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. COMPSTAT, 2010.

[4] Richard H Byrd, Gillian M Chin, Jorge Nocedal, Yuchen Wu: “Sample size selection in optimization methods for machine learning”. Mathematical programming 134(1):127-155, 2012.

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers” Foundations and Trends in Machine Learning, 3(1):1–122, 2011.

[6] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM : a library for support vector machines”. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[7] John C. Duchi, Alekh Agarwal, and Martin J. Wainwright. “Dual averaging for distributed optimization”. IEEE Transactions on Automatic Control, 57(3):592–606, 2012

[8] Michael P. Friedlander and Mark Schmidt. “Hybrid deterministic-stochastic methods for data fitting”. SIAM Journal on Scientific Computing, 34(3):A1380–A1405, 2012.

[9] Bingsheng He and Xiaoming Yuan. “On the $\mathbb{O}$(1/t) convergence rate of the douglas-rachford alternating direction method”. SIAM Journal on Numerical Analysis, 50(2):700:709, 2012.

[10] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan: “A dual coordinate descent method for large-scale linear SVM”. ICML, 2008.

[11] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. “Group lasso with overlap”. and graph lasso. ICML, 2009.

[12] Jorge Nocedal, Stephen J Wright. “Numerical optimization“ Springer New York, 2006.

[13] Martin Takáč, Avleen Bijral, Peter Richtrik, and Nathan Srebro. “Mini-batch primal and acdual methods for svms”. ICML, 2013.

[14] Hua Ouyang, Niao He, Long Q. Tran, Alexander Gray. “Stochastic Alternating Direction Method of Multiplier” ICML, 2013.

[15] Shai Shalev-Shwartz and Tong Zhang. “Stochastic dual coordinate ascent methods for reg- ularized loss minimization.” Journal of Machine Learning Research, 14(Feb):567599, 2013a

[16] Shai Shalev-Shwartz and Tong Zhang. “Proximal stochastic dual coordinate ascent”. arXiv:1211.2717, 2013b.

[17] Tajii Suzuki. “Dual averaging and proximal gradient descent for online alternating direction multiplier method.” ICML, 2013.

[18] Tajii Suzuki. “Stochastic dual coordinate ascent with alternating direction method of multipliers”. ICML, 2014.

[19] Antonio Torralba and Alexei A. Efros. “Unbiased look at dataset bias”. ICCV, 2011.

[20] Huahua Wang and Arindam Banerjee. “Online Alternating Direction Method” ICML, 2012.

[21] Lin Xiao. “Dual averaging methods for regularized stochastic learning and online optimization”. Journal of Machine Learning Research. 11:25432596, 2010.

[22] Tianbao Yang “Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent”. NIPS, 2013

[23] Martin Zinkevich, Markus Weimer, Alex Smola, Lihong Li. “Parallelized Stochastic Gradient Descent”. NIPS, 2010.

[24] Caoxie Zhang, Honglak Lee, and Kang G. Shin. “Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers”. AISTATS, 2012.

[25] Ruiliang Zhang and James. Kwok. “Asynchronous distributed ADMM for consensus optimization”. ICML, 2014.

[26] Wenliang Zhong and James Tin-Yau Kwok “Fast stochastic alternating direction method of multipliers.” ICML, 2014.

[27] Yuchen Zhang, John Duchi, Michael I. Jordan, Martin Wainwright. “Information-theoretic Lower Bounds for Distributed Statistical Estimation with Communication Constraints”. NIPS, 2013.

Appendix

Here, we give a complete proof for the theorem. It basically follows the standard analysis of ADMM as [20] except the additional deviation term $e^{(t)}$.

Proof: By the partial optimality of $L_\rho(w, z, \lambda)$ with respect to w, we have

$$\partial f(w^{(t+1)}) + e^{(t)} + A^\top(\lambda^{(t)} + \rho(Aw^{(t+1)} + Bz^{(t)} - c)) = 0. \tag{46}$$

Using the update rule

$$\lambda^{(t+1)} = \lambda^{(t)} + \rho(Aw^{(t+1)} + Bz^{(t+1)} - c), \tag{47}$$

yields

$$\partial f(w^{(t+1)}) + e^{(t)} + A^\top(\lambda^{(t+1)} + \rho(Bz^{(t)} - Bz^{(t+1)})) = 0. \tag{48}$$

Similarly, for z, we have

$$\partial g(z^{(t+1)}) + B^\top(\lambda^{(t)} + \rho(Aw^{(t+1)} + Bz^{(t+1)} - c)) = 0. \tag{49}$$

Using the update rule (47) yields

$$\partial g(z^{(t+1)}) + B^\top\lambda^{(t+1)} = 0. \tag{50}$$

By the convexity of f,

$$\begin{aligned}
&f(w^{(t+1)}) - f(w^*) \leq -\langle e^{(t)} + A^\top(\lambda^{(t+1)} + \rho(Bz^{(t)} - Bz^{(t+1)}))\\
&, w^{(t+1)} - w^*\rangle = -\langle \lambda^{(t+1)} + \rho(Bz^{(t)} - Bz^{(t+1)}), Aw^{(t+1)} - Aw^*\rangle\\
&- \langle e^{(t)}, w^{(t+1)} - w^*\rangle\\
&= -\langle \lambda^{(t+1)}, Aw^{(t+1)} + Bz^* - c\rangle + \rho\langle (Bz^{(t+1)} - Bz^{(t)}), Aw^{(t+1)}\\
&+ Bz^* - c\rangle - \langle e^{(t)}, w^{(t+1)} - w^*\rangle\\
&= -\langle \lambda^{(t+1)}, Aw^{(t+1)} + Bz^* - c\rangle + \frac{\rho}{2}(\|Bz^* - Bz^{(t)}\|_2^2 - \|Bz^*\\
&- Bz^{(t+1)}\|_2^2 + \|Aw^{(t+1)} + Bz^{(t+1)} - c\|_2^2 - \|Aw^{(t+1)} + Bz^{(t)} -\\
&c\|_2^2) - \langle e^{(t)}, w^{(t+1)} - w^*\rangle.
\end{aligned} \tag{51}$$

Similarly, by the convexity of g(z),

$$g(z^{(t+1)}) - g(z^*) \leq -\langle B^\top\lambda^{(t+1)}, z^{(t+1)} - z^*\rangle. \tag{52}$$

Add (51) and (52), we have

$$\begin{aligned}
&f(w^{(t+1)}) + g(z^{(t+1)}) - f(w^*) - g(z^*) \leq\\
&- \langle \lambda^{(t+1)}, Aw^{(t+1)} + Bz^{(t+1)} - c\rangle - \langle e^{(t)}, w^{(t+1)} - w^*\rangle + \frac{\rho}{2}(\|B\\
&z^* - Bz^{(t)}\|_2^2 - \|Bz^* - Bz^{(t+1)}\|_2^2 + \|Aw^{(t+1)} + Bz^{(t+1)} - c\|_2^2 -\\
&\|Aw^{(t+1)} + Bz^{(t)} - c\|_2^2)\\
&= -\langle e^{(t)}, w^{(t+1)} - w^*\rangle + \frac{\rho}{2}(\|Bz^* - Bz^{(t)}\|_2^2 - \|Bz^* - Bz^{(t+1)}\\
&\|_2^2 - \|Aw^{(t+1)} + Bz^{(t)} - c\|_2^2) + \frac{1}{2\rho}(\|\lambda^{(t)}\|_2^2 - \|\lambda^{(t+1)}\|_2^2)\\
&\leq -\langle e^{(t)}, w^{(t+1)} - w^*\rangle + \frac{\rho}{2}(\|Bz^* - Bz^{(t)}\|_2^2 - \|Bz^* - Bz^{(t+1)}\\
&\|_2^2) + \frac{1}{2\rho}(\|\lambda^{(t)}\|_2^2 - \|\lambda^{(t+1)}\|_2^2).
\end{aligned} \tag{53}$$

Summing the above inequality from $t = 0$ to $T - 1$, we have

$$\begin{aligned}
&\Sigma_{t=0}^{T-1} f(w^{(t+1)}) + g(z^{(t+1)}) - f(w^*) - g(z^*)\\
&\leq \frac{\rho}{2}(\|Bz^* - Bz^{(0)}\|_2^2 - \|Bz^* - Bz^{(T)}\|_2^2) + \frac{1}{2\rho}(\|\lambda^{(0)}\|_2^2 -\\
&\|\lambda^{(T)}\|_2^2) - \Sigma_{t=0}^{T-1}\langle e^{(t)}, w^{(t+1)} - w^*\rangle\\
&\leq \frac{\rho}{2}(\|Bz^* - Bz^{(0)}\|_2^2 - \|Bz^* - Bz^{(T)}\|_2^2) + \frac{1}{2\rho}(\|\lambda_0\|_2^2 -\\
&\|\lambda_T\|_2^2) + \Sigma_{t=0}^{T-1}\|e^{(t)}\|\|w^{(t+1)} - w^*\|.
\end{aligned} \tag{54}$$

Using Jensen’s inequality yields

$$\begin{aligned}
&f(\overline{w}_T) + g(\overline{z}_T) - f(w^*) - g(z^*)\\
&\leq \frac{\rho}{2T}(\|Bz^* - Bz^{(0)}\|_2^2 - \|Bz^* - Bz^{(T)}\|_2^2) +\\
&\frac{1}{2\rho T}(\|\lambda^{(0)}\|_2^2 - \|\lambda^{(T)}\|_2^2) + \frac{1}{T}\Sigma_{t=0}^{T-1}\|e^{(t)}\|\|w^{(t+1)} - w^*\|.
\end{aligned} \tag{55}$$

■