On Simplifying the Primal-Dual Method of Multipliers

Zhang, Guoqiang; Heusdens, Richard

DOI
10.1109/icassp.2016.7472594

Publication date
2016

Document Version
Accepted author manuscript

Published in
2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)

ON SIMPLIFYING THE PRIMAL-DUAL METHOD OF MULTIPLIERS

Guoqiang Zhang and Richard Heusdens

Circuits and System group
Delft University of Technology, the Netherlands
{g.zhang-1,r.heusdens}@tudelft.nl

ABSTRACT

Recently, the primal-dual method of multipliers (PDMM) has been proposed to solve a convex optimization problem defined over a general graph. In this paper, we consider simplifying PDMM for a subclass of the convex optimization problems. This subclass includes the consensus problem as a special form. By using algebra, we show that the update expressions of PDMM can be simplified significantly. We then evaluate PDMM for training a support vector machine (SVM). The experimental results indicate that PDMM converges considerably faster than the alternating direction method of multipliers (ADMM).

Index Terms— Distributed optimization, PDMM, ADMM, SVM.

1. INTRODUCTION

In recent years, distributed optimization has attracted increasing attention driven by two main motivations. Firstly, various types of networks are invented and employed for collecting data, monitoring the environment, managing facilities such as wireless sensor networks, smart grid and Internet of things. In the above situation, distributed optimization is desirable to perform network resource allocation, utility maximization and as such. Secondly, processing of big data usually requires many computing units (e.g., a computer or a GPU) to work jointly, where each unit processes a portion of the data. Distributed optimization is then required for coordination of the computing units [1].

In the last decade, various methods have been proposed for distributed optimization. The alternating-direction method of multipliers (ADMM) is probably the most popular algorithm being applied in practice (see [2] for an overview of the applications). Specifically, ADMM intends to solve the following convex optimization problem in a distributed manner

$$\min_{\boldsymbol{x},\boldsymbol{z}} f(\boldsymbol{x}) + g(\boldsymbol{z}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}, \tag{1}$$

where $f(\cdot)$ and $g(\cdot)$ are two convex functions. The two matrices $(\boldsymbol{A}, \boldsymbol{B})$ and the vector $\boldsymbol{c}$ are properly set to be in line with the dimensions of $\boldsymbol{x}$ and $\boldsymbol{z}$. Problem (1) can be considered to be defined over a graph with two nodes where each node carries either $f(\cdot)$ or $g(\cdot)$. Recently, ADMM has also been extended to solve nonconvex optimization problems [3].

One limitation with ADMM is that the method is applicable only when a distributed optimization problem can be formulated into (1). In some situations, the problem formulation may have to introduce quite a few auxiliary variables, making the method less efficient.

This work was supported by the COMMIT program, The Netherlands.

Recently, we have proposed a new algorithm named *primal-dual method of multipliers* (PDMM)[1] for solving a convex optimization problem defined over a general graph $G = (\mathcal{V}, \mathcal{E})$ (see [4, 5])

$$\min_{\boldsymbol{x}} \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i) \text{ s. t. } \boldsymbol{A}_{ij}\boldsymbol{x}_i + \boldsymbol{A}_{ji}\boldsymbol{x}_j = \boldsymbol{c}_{ij}\ \forall (i,j) \in \mathcal{E}, \tag{2}$$

where every node $i \in \mathcal{V}$ carries a convex function $f_i(\cdot)$, and every edge $(i, j) \in \mathcal{E}$ carries an equality constraint $\boldsymbol{A}_{ij}\boldsymbol{x}_i + \boldsymbol{A}_{ji}\boldsymbol{x}_j = \boldsymbol{c}_{ij}$. PDMM can be taken as an extension of ADMM for solving problems over general graphs. We note that Problem (2) can also be solved by ADMM by reformulating (2) into (1). An empirical study in [5] indicates PDMM converges considerably faster than ADMM for the distributed averaging problem (see [6] for the pioneering work).

This paper presents two new contributions. Firstly, we consider a subclass of Problem (2), which takes the form of

$$\min_{\boldsymbol{x}} \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i) \text{ s. t. } \boldsymbol{B}_{ij}\boldsymbol{x}_i = \boldsymbol{B}_{ji}\boldsymbol{x}_j \quad \forall (i,j) \in \mathcal{E}, \tag{3}$$

We show that the updating expressions of PDMM for the subclass (3) can be simplified considerably, making it more attractive for practical usage. The subclass (3) includes the consensus problem as a special case, where every edge (i, j) carries an equality contraint $\boldsymbol{x}_i = \boldsymbol{x}_j$.

Secondly, we apply PDMM to train a support vector machine (SVM), where the training samples are distributed across a set of computing units. Every unit can communicate with all the other units at each iteration. In other words, the set of computing units form a fully connected graph. Experimental results demonstrate that PDMM not only converges considerably faster but also is less sensitive to the parameter selection than ADMM w.r.t. the convergence speed.

2. PROBLEM FORMULATION

Considering the problem (3), we let $(\boldsymbol{B}_{ij}, \boldsymbol{B}_{ji}) \in (\mathbb{R}^{n_{ij}\times n_i}, \mathbb{R}^{n_{ij}\times n_j})$ for every edge $(i, j) \in \mathcal{E}$. We use $\mathcal{N}_i$ to denote the set of neighbors of node i and $\mathcal{V} = \{1, 2, \ldots, m\}$ to denote the vertex set (set of all nodes in the graph). As a result, $|\mathcal{V}| = m$. In order to make use of PDMM in [5], we reformulate (3) into the form of (2)

$$\begin{aligned} &\min_{\boldsymbol{x}} \sum_{i\in\mathcal{V}} f_i(\boldsymbol{x}_i) \\ &\text{s. t. } \mathrm{u}_{i-j}\boldsymbol{B}_{ij}\boldsymbol{x}_i + \mathrm{u}_{j-i}\boldsymbol{B}_{ji}\boldsymbol{x}_j = \boldsymbol{0} \quad \forall (i,j) \in \mathcal{E}, \end{aligned} \tag{4}$$

[1]The algorithm is originally named as the bi-alternating direction method of multipliers (BiADMM), but later on changed to PDMM.

where $u_{(\cdot)}$ is a sign function defined as

$$u_y = \begin{cases} 1 & y > 0 \\ -1 & y < 0 \end{cases}. \tag{5}$$

In (4), u_{i-j} and u_{j-i} always have opposite signs, i.e., $u_{i-j} \cdot u_{j-i} = -1$.

Given the primal problem (4), the Lagrangian function can then be constructed as

$$L(\boldsymbol{x}, \boldsymbol{\delta}) = \sum_{i \in \mathcal{V}} f_i(\boldsymbol{x}_i) - \sum_{(i,j) \in \mathcal{E}} \boldsymbol{\delta}_{ij}^T (u_{i-j} \boldsymbol{B}_{ij} \boldsymbol{x}_i + u_{j-i} \boldsymbol{B}_{ji} \boldsymbol{x}_j), \tag{6}$$

where $\boldsymbol{\delta}_{ij}$ is the Lagrangian multiplier (or the dual variable) for each constraint $u_{i-j}\boldsymbol{B}_{ij}\boldsymbol{x}_i + u_{j-i}\boldsymbol{B}_{ji}\boldsymbol{x}_j = \boldsymbol{0}$ in (4). The vector $\boldsymbol{\delta}$ is obtained by stacking the individual variables $\boldsymbol{\delta}_{ij}$, $(i,j) \in \mathcal{E}$. Therefore, $\boldsymbol{x} \in \mathbb{R}^{\sum_i n_i}$ and $\boldsymbol{\delta} \in \mathbb{R}^{\sum_{(i,j)} n_{ij}}$. The Lagrangian function is convex in $\boldsymbol{x}$ for fixed $\boldsymbol{\delta}$, and concave in $\boldsymbol{\delta}$ for fixed $\boldsymbol{x}$. Throughout the rest of the paper, we will make the following (common) assumption:

Assumption 1. *There exists a saddle point $(\boldsymbol{x}^*, \boldsymbol{\delta}^*)$ to the Lagrangian function $L(\boldsymbol{x}, \boldsymbol{\delta})$ such that for all $\boldsymbol{x} \in \mathbb{R}^{\sum_i n_i}$ and $\boldsymbol{\delta} \in \mathbb{R}^{\sum_{(i,j)} n_{ij}}$ we have*

$$L(\boldsymbol{x}^*, \boldsymbol{\delta}) \le L(\boldsymbol{x}^*, \boldsymbol{\delta}^*) \le L(\boldsymbol{x}, \boldsymbol{\delta}^*).$$

3. SIMPLIFYING THE PRIMAL-DUAL METHOD OF MULTIPLIERS

In this section, we first briefly introduce PDMM for solving (4). We note that every matrix $\boldsymbol{B}_{ij}$ is coupled with the sign function u_{i-j} in (4). As a result, the function $u_{(\cdot)}$ also appears in the updating expressions of PDMM, making the implementation of the algorithm a bit difficult.

We show that by using algebra, the sign function $u_{(\cdot)}$ can be removed from the updating expressions. After simplifying the updating expressions, there is no need to track the sign function $u_{(\cdot)}$ when implementing PDMM.

3.1. The updating expressions of PDMM

PDMM iteratively optimizes an augmented primal-dual Lagrangian function to approach the optimal solution of (4) (see [5]). Before presenting the function, we first introduce a few auxiliary variables. We let $\boldsymbol{\lambda}_{i|j}$ and $\boldsymbol{\lambda}_{j|i}$ be two (dual) variables for every edge $(i,j) \in \mathcal{E}$, which are of the same dimension as $\boldsymbol{\delta}_{ij}$ in (6). The variable $\boldsymbol{\lambda}_{i|j}$ is owned by and updated at node i and is related to neighboring node j. We use $\boldsymbol{\lambda}_i$ to denote the vector by concatenating all $\boldsymbol{\lambda}_{i|j}$, $j \in \mathcal{N}_i$. Finally, we let $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1^T, \ldots, \boldsymbol{\lambda}_{|\mathcal{V}|}^T]^T$.

Upon introducing $\boldsymbol{\lambda}$, the augmented primal-dual Lagrangian function for (4) can be expressed as (see [5])

$$L_{\mathcal{P}}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i \in \mathcal{V}} \Big[f_i(\boldsymbol{x}_i) - \sum_{j \in \mathcal{N}(i)} \boldsymbol{\lambda}_{j|i}^T (u_{i-j} \boldsymbol{B}_{ij} \boldsymbol{x}_i) - f_i^*\Big(\sum_{j \in \mathcal{N}_i} u_{i-j} \boldsymbol{B}_{ij}^T \boldsymbol{\lambda}_{i|j} \Big) \Big] + h_{\mathcal{P}}(\boldsymbol{x}) - g_{\mathcal{P}}(\boldsymbol{\lambda}) \tag{7}$$

where f_i^* is the conjugate function (see [7] for the definition) of f_i, and $h_{\mathcal{P}}(\boldsymbol{x})$ and $g_{\mathcal{P}}(\boldsymbol{\lambda})$ are defined as

$$h_{\mathcal{P}}(\boldsymbol{x}) = \sum_{(i,j) \in \mathcal{E}} \frac{1}{2} \left\| u_{i-j} \boldsymbol{B}_{ij} \boldsymbol{x}_i + u_{j-i} \boldsymbol{B}_{ji} \boldsymbol{x}_j \right\|_{\boldsymbol{P}_{ij}}^2 \tag{8}$$

$$g_{\mathcal{P}}(\boldsymbol{\lambda}) = \sum_{(i,j) \in \mathcal{E}} \frac{1}{2} \left\| \boldsymbol{\lambda}_{i|j} - \boldsymbol{\lambda}_{j|i} \right\|_{\boldsymbol{P}_{ij}^{-1}}^2, \tag{9}$$

where $\mathcal{P} = \{\boldsymbol{P}_{ij} = \boldsymbol{P}_{ij}^T \succ 0 | (i,j) \in \mathcal{E}\}$ is a set of positive definite matrices to be specified. $L_{\mathcal{P}}$ is convex in $\boldsymbol{x}$ for fixed $\boldsymbol{\lambda}$ and concave in $\boldsymbol{\lambda}$ for fixed $\boldsymbol{x}$.

It is shown in [5] that instead of solving the original problem (4), one can alternatively find a saddle point of the function $L_{\mathcal{P}}$. At each iteration, PDMM iteratively optimizes $L_{\mathcal{P}}$ to obtain a new estimate $(\boldsymbol{x}^{k+1}, \boldsymbol{\lambda}^{k+1})$ based on $(\boldsymbol{x}^k, \boldsymbol{\lambda}^k)$ obtained from the $k-1$th iteration, where $k \ge 1$. Specifically, the new estimate $(\boldsymbol{x}^{k+1}, \boldsymbol{\lambda}^{k+1})$ is computed as

$$\Big(\boldsymbol{x}_i^{k+1}, \boldsymbol{\lambda}_i^{k+1}\Big) = \arg\min_{\boldsymbol{x}_i} \max_{\boldsymbol{\lambda}_i} L_{\mathcal{P}}\Big(\Big[\ldots, \boldsymbol{x}_{i-1}^{k,T}, \boldsymbol{x}_i^T, \boldsymbol{x}_{i+1}^{k,T}, \ldots\Big]^T, \Big[\ldots, \boldsymbol{\lambda}_{i-1}^{k,T}, \boldsymbol{\lambda}_i^T, \boldsymbol{\lambda}_{i+1}^{k,T}, \ldots\Big]^T \Big) \quad i \in \mathcal{V}. \tag{10}$$

Combining (7) and (10) produces the updating expressions which are summarized in Table 1.

Initialization: Randomly initialize $\{\boldsymbol{x}_i\}$ and $\{\boldsymbol{\lambda}_{i|j}\}$
Repeat
  for all $i \in \mathcal{V}$ do
    $\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i} \Big[f_i(\boldsymbol{x}_i) - \boldsymbol{x}_i^T (\sum_{j \in \mathcal{N}_i} u_{i-j} \boldsymbol{B}_{ij}^T \boldsymbol{\lambda}_{j|i}^k) + \sum_{j \in \mathcal{N}_i} \frac{1}{2} \| u_{i-j} \boldsymbol{B}_{ij} \boldsymbol{x}_i + u_{j-i} \boldsymbol{B}_{ji} \boldsymbol{x}_j^k \|_{\boldsymbol{P}_{ij}}^2 \Big]$
  end for
  for all $i \in \mathcal{V}$ and $j \in \mathcal{N}_i$ do
    $\boldsymbol{\lambda}_{i|j}^{k+1} = \boldsymbol{\lambda}_{j|i}^k - \boldsymbol{P}_{ij} (u_{j-i} \boldsymbol{B}_{ji} \boldsymbol{x}_j^k + u_{i-j} \boldsymbol{B}_{ij} \boldsymbol{x}_i^{k+1})$
  end for
  $k \leftarrow k+1$
Until some stopping criterion is met

Table 1. Procedure of PDMM

3.2. Expression simplification

We note that in (7)-(9) and Table 1, the sign function $u_{(\cdot)}$ is heavily involved, which complicates the implementation of PDMM. We will show in the following that by proper variable replacement, the sign function $u_{(\cdot)}$ can be removed from the updating expressions.

We introduce a new variable $\boldsymbol{\beta}_{i|j}$ to replace the variable $\boldsymbol{\lambda}_{i|j}$, which is defined as

$$\boldsymbol{\lambda}_{i|j} = u_{i-j} \boldsymbol{\beta}_{i|j} \quad \forall i \in \mathcal{V}, j \in \mathcal{N}_i. \tag{11}$$

We use $\boldsymbol{\beta}_i$ to denote the vector by concatenating $\boldsymbol{\beta}_{i|j}$, $j \in \mathcal{N}_i$. Finally, we let $\boldsymbol{\beta} = [\boldsymbol{\beta}_1^T, \ldots, \boldsymbol{\beta}_{|\mathcal{V}|}^T]^T$.

We now simplify (7)-(9) with the vector $\boldsymbol{\beta}$. We start with the function $g_{\mathcal{P}}(\boldsymbol{\lambda})$. Plugging (11) into $g_{\mathcal{P}}(\boldsymbol{\lambda})$ produces

$$g_{\mathcal{P}}(\boldsymbol{\lambda}(\boldsymbol{\beta})) = \sum_{(i,j) \in \mathcal{E}} \frac{1}{2} \left\| u_{i-j} \boldsymbol{\beta}_{i|j} - u_{j-i} \boldsymbol{\beta}_{j|i} \right\|_{\boldsymbol{P}_{ij}^{-1}}^2 \overset{(a)}{=} \sum_{(i,j) \in \mathcal{E}} \frac{1}{2} \left\| \boldsymbol{\beta}_{i|j} + \boldsymbol{\beta}_{j|i} \right\|_{\boldsymbol{P}_{ij}^{-1}}^2, \tag{12}$$

Initialization: Randomly initialize $\{\boldsymbol{x}_i\}$ and $\{\boldsymbol{\beta}_{i|j}\}$
Repeat
  for all $i \in \mathcal{V}$ do
    $\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i}\Big[f_i(\boldsymbol{x}_i) + \boldsymbol{x}_i^T(\sum_{j\in\mathcal{N}_i}\boldsymbol{B}_{ij}^T\boldsymbol{\beta}_{j|i}^k) + \sum_{j\in\mathcal{N}_i}\frac{1}{2}\|\boldsymbol{B}_{ij}\boldsymbol{x}_i - \boldsymbol{B}_{ji}\boldsymbol{x}_j^k\|_{\boldsymbol{P}_{ij}}^2\Big]$
  end for
  for all $i \in \mathcal{V}$ and $j \in \mathcal{N}_i$ do
    $\boldsymbol{\beta}_{i|j}^{k+1} = -\boldsymbol{\beta}_{j|i}^k + \boldsymbol{P}_{ij}(\boldsymbol{B}_{ji}\boldsymbol{x}_j^k - \boldsymbol{B}_{ij}\boldsymbol{x}_i^{k+1})$
  end for
  $k \leftarrow k+1$
Until some stopping criterion is met

Table 2. Procedure of the simplified PDMM

where in (12), we use the property that u_{i-j} and u_{j-i} always have the opposite signs. Similarly, the two functions $h_{\mathcal{P}}(\boldsymbol{x})$ and $L_{\mathcal{P}}(\boldsymbol{x},\boldsymbol{\lambda})$ can be simplified in terms of $\boldsymbol{x}$ and $\boldsymbol{\beta}$ as

$$L_{\mathcal{P}}(\boldsymbol{x},\boldsymbol{\lambda}(\boldsymbol{\beta})) = \sum_{i\in\mathcal{V}}\Big[f_i(\boldsymbol{x}_i) + \sum_{j\in\mathcal{N}(i)}\boldsymbol{\beta}_{j|i}^T(\boldsymbol{B}_{ij}\boldsymbol{x}_i) - f_i^*\Big(\sum_{j\in\mathcal{N}_i}\boldsymbol{B}_{ij}^T\boldsymbol{\beta}_{i|j}\Big)\Big] + h_{\mathcal{P}}(\boldsymbol{x}) - g_{\mathcal{P}}(\boldsymbol{\lambda}(\boldsymbol{\beta})) \quad (13)$$

$$h_{\mathcal{P}}(\boldsymbol{x}) = \sum_{(i,j)\in\mathcal{E}}\frac{1}{2}\|\boldsymbol{B}_{ij}\boldsymbol{x}_i - \boldsymbol{B}_{ji}\boldsymbol{x}_j\|_{\boldsymbol{P}_{ij}}^2. \quad (14)$$

Finally combining (12)-(14) and (10) produces the updating expressions shown in Table 2.

Remark 1. *We note that due to limited space, we have only described the synchronous PDMM (i.e., all the variables are updated simultaneously). The derivation above in fact also holds for the asynchronous PDMM (i.e., a portion of variables are updated at each iteration).*

4. SVM TRAINING

In this section, we consider training an SVM by using both PDMM and ADMM. We assume that the training data are distributed across a set of computing units, e.g., computers. The set of computing units can communicate with each other directly, which can be modeled as a fully connected graph (one node for each computing unit). We will show that PDMM is considerably more efficient than ADMM for training the SVM on the fully connected graph.

4.1. Problem formulation

For simplification, we consider training an SVM for two classes by finding the hyperplane $(\boldsymbol{w}, b)$ between them [8], where $\boldsymbol{w}$ is the norm of the hyperplane and b is the offset. We denote the fully connected graph as $G_f = (\mathcal{V}, \mathcal{E}_f)$, where $\mathcal{E}_f = \{(i,j)|i,j \in \mathcal{V}, i \neq j\}$. Each node $i \in \mathcal{V}$ has l_i pairs of training samples and labels $(\boldsymbol{z}_i^t, y_i^t)$, $t = 1,\ldots,l_i$. The label y_i^t either equals to -1 or 1 depending on which class the training sample $\boldsymbol{z}_i^t$ belongs to. Further, we assume that every node $i \in \mathcal{V}$ carries a copy $(\boldsymbol{w}_i, b_i)$ of the hyperplane $(\boldsymbol{w}, b)$.

Upon introducing the above notations, the training for the SVM over the graph $G_f = (\mathcal{V}, \mathcal{E}_f)$ can be formulated as

$$\min\sum_{i\in\mathcal{V}} f_i(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i) \text{ s.t. } (\boldsymbol{w}_i, b_i) = (\boldsymbol{w}_j, b_j), \forall (i,j) \in \mathcal{E}_f, \quad (15)$$

where each function f_i, $i \in \mathcal{V}$, is given by

$$f_i(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i) = \frac{1}{2}\|\boldsymbol{w}_i\|^2 + C\sum_{t=1}^{l_i}\xi_i^t \quad (16)$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}_i^T\boldsymbol{z}_i^t + b_i) \geq 1 - \xi_i^t \quad t = 1,\ldots,l_i \quad (17)$$

$$\xi_i^t \geq 0 \qquad t = 1,\ldots,l_i, \quad (18)$$

where $\boldsymbol{\xi}_i = [\xi_i^1, \xi_i^2, \ldots, \xi_i^{l_i}]^T$ and C is a constant. The minimization in (15) is over all the variables $\{(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i)|i \in \mathcal{V}\}$. The research goal is for the graph G_f to perform distributed optimization to reach a consensus of the optimal hyperplane $(\boldsymbol{w}_i, b_i) = (\boldsymbol{w}^*, b^*)$, $i \in \mathcal{V}$, where $(\boldsymbol{w}^*, b^*)$ is the optimal solution.

4.2. Training by PDMM

In this subsection, we consider applying PDMM to solve the training problem (15). To be able to convert the problem into (12)-(14), we let $\boldsymbol{x}_i = [\boldsymbol{w}_i^T, b_i, \boldsymbol{\xi}_i^T]^T$ and $\boldsymbol{B}_{ij}\boldsymbol{x}_i = [\boldsymbol{w}_i^T, b_i]^T$ for all $j \in \mathcal{N}_i$. As a result, the function $h_{\mathcal{P}}(\boldsymbol{x})$ becomes

$$h_{\mathcal{P}}(\{\boldsymbol{w}_i, b_i\}) = \sum_{i\neq j}\frac{1}{2}\left\|\begin{pmatrix}\boldsymbol{w}_i\\ b_i\end{pmatrix} - \begin{pmatrix}\boldsymbol{w}_j\\ b_j\end{pmatrix}\right\|_{\boldsymbol{P}_{ij}}^2 \quad (19)$$

To simplify the computation later on, we choose the set $\mathcal{P}$ such that

$$h_{\mathcal{P}}(\{\boldsymbol{w}_i, b_i\}) = \sum_{j\neq i}\Big(\frac{\gamma}{2}\left\|\boldsymbol{w}_i - \boldsymbol{w}_j^k\right\|^2 + \frac{\gamma(m-1)+1}{2(m-1)}(b_i - b_j^k)^2\Big), \quad (20)$$

where m represents the number of nodes in the graph, and $\gamma > 0$ which characterizes all the $\boldsymbol{P}_{ij}$ matrices. One can also work out the expressions for $L_{\mathcal{P}}(\boldsymbol{x},\boldsymbol{\lambda}(\boldsymbol{\beta}))$ and $g_{\mathcal{P}}(\boldsymbol{\lambda}(\boldsymbol{\beta}))$ in a similar manner.

We now derive the updating expression for $\{\boldsymbol{w}_i^{k+1}, b_i^{k+1}\}$ given the estimate $\{\boldsymbol{w}_i^k, b_i^k\}$ at iteration k. By plugging (16), (20) and $\boldsymbol{B}_{ij}\boldsymbol{x}_i = [\boldsymbol{w}_i^T, b_i]^T$ into the algorithm described in Table 2, the new estimate $(\boldsymbol{w}_i^{k+1}, b_i^{k+1})$ can be computed as

$$\begin{aligned}&(\boldsymbol{w}_i^{k+1}, b_i^{k+1}, \boldsymbol{\xi}_i^{k+1})\\ &= \arg\min\Bigg[\frac{1}{2}\|\boldsymbol{w}_i\|^2 + C\sum_{t=1}^{l_i}\xi_i^t + \begin{pmatrix}\boldsymbol{w}_i\\ b_i\end{pmatrix}^T\Big(\sum_{j\neq i}\boldsymbol{\beta}_{j|i}^k\Big)\\ &\quad + \sum_{j\neq i}\Big(\frac{\gamma}{2}\left\|\boldsymbol{w}_i - \boldsymbol{w}_j^k\right\|^2 + \frac{\gamma(m-1)+1}{2(m-1)}(b_i - b_j^k)^2\Big)\Bigg] \quad i \in \mathcal{V}, \quad (21)\end{aligned}$$

where $(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i)$ satisfies the inequality constraints (17)-(18).

Finally by using the duality concept [7], the problem (21) can be reformulated as

$$\begin{pmatrix}\boldsymbol{w}_i^{k+1}\\ b_i^{k+1}\end{pmatrix} = \frac{1}{(1+(m-1)\gamma)}\Bigg[\sum_{t=1}^{l_i}\alpha_i^{t,k+1}y_i^t\begin{pmatrix}\boldsymbol{z}_i^t\\ 1\end{pmatrix} + \sum_{j\neq i}\Bigg(\begin{pmatrix}\gamma\boldsymbol{w}_j^k\\ \frac{\gamma(m-1)+1}{m-1}b_j^k\end{pmatrix} - \boldsymbol{\beta}_{j|i}^k\Bigg)\Bigg] \quad i \in \mathcal{V},$$

where $\boldsymbol{\alpha}_i^{k+1} = [\alpha_i^{1,k+1}, \ldots, \alpha_i^{t,k+1}]^T$ is computed as

$$\boldsymbol{\alpha}_i^{k+1} = \arg\max_{\boldsymbol{\alpha}_i} \left[\sum_{t=1}^{l_i} \alpha_i^t - \frac{1}{2(1+(m-1)\gamma)} \left\| \sum_{t=1}^{l_i} \alpha_i^t y_i^t \begin{pmatrix} \boldsymbol{z}_i^t \\ 1 \end{pmatrix} + \sum_{j \neq i} \left(\left(\frac{\gamma \boldsymbol{w}_j^k}{\frac{\gamma(m-1)+1}{m-1}} b_j^k \right) - \boldsymbol{\beta}_{j|i}^k \right) \right\|^2 \right] \quad i \in \mathcal{V}, \quad (22)$$

where $C \geq \alpha_i^t \geq 0$ for all $t = 1, \ldots, l_i, i \in \mathcal{V}$.

4.3. Training by ADMM

In this subsection, we briefly explain how to explore ADMM for training the SVM distributively. The basic idea is to reformulate (15) into the form of (1). To do so, we introduce a new function $g(\boldsymbol{w}, b) = 0$. As a result, (15) can be reformulated as

$$\min \sum_{i \in \mathcal{V}} f_i(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i) + g(\boldsymbol{w}, b) \text{ s.t. } (\boldsymbol{w}_i, b_i) = (\boldsymbol{w}, b) \ \forall i \in \mathcal{V}. \quad (23)$$

The ADMM then intends to optimize an augmented Lagrangian function iteratively which is built on (23). The function can be expressed as [2]

$$\begin{aligned} & L_\rho(\{\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i\}, (\boldsymbol{w}, b), \boldsymbol{r}) \\ &= \sum_{i \in \mathcal{V}} f_i(\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i) + g(\boldsymbol{w}, b) + \sum_{i \in \mathcal{V}} \boldsymbol{r}_i^T \begin{pmatrix} \boldsymbol{w}_i - \boldsymbol{w} \\ b_i - b \end{pmatrix} \\ &\quad + \sum_{i \in \mathcal{V}} \frac{\rho}{2} \left\| \begin{pmatrix} \boldsymbol{w}_i \\ b_i \end{pmatrix} - \begin{pmatrix} \boldsymbol{w} \\ b \end{pmatrix} \right\|^2, \end{aligned} \quad (24)$$

where $\rho > 0$ and $\boldsymbol{r} = [\boldsymbol{r}_1^T, \ldots, \boldsymbol{r}_m^T]^T$ is the Lagrangian multiplier. At each iteration, the three sets of variables $\{\boldsymbol{w}_i, b_i, \boldsymbol{\xi}_i\}$, $(\boldsymbol{w}, b)$ and $\boldsymbol{r}$ are sequentially updated one after another (see [2]).

4.4. Experimental results

In the experiment, we evaluated both PDMM and ADMM in terms of the convergence speed. The number of nodes in G_f was set as $m = |\mathcal{V}| = 3$. The training samples for the two classes were randomly generated in a 2-dimensional space (See Fig 1:(a)). The SVM training is to find a line that well separates the two class of samples. In total, there are 1200 training samples, where each class has 600 samples. The training samples are evenly distributed over the 3 nodes in the graph. The parameter C in (16) was set as $C = 1$.

To make a fair comparison between the two algorithms, we first utilized all the training samples to compute a global solution $(\boldsymbol{w}^{\text{glob}}, b^{\text{glob}})$ (corresponding to the line in Fig 1:(a)). When implimenting the two algorithms, we chose the error criterion at each iteration to be

$$\text{error}^k = \left\| \frac{1}{m} \sum_{i \in \mathcal{V}} \begin{pmatrix} \boldsymbol{w}_i^k \\ b_i^k \end{pmatrix} - \begin{pmatrix} \boldsymbol{w}^{\text{glob}} \\ b^{\text{glob}} \end{pmatrix} \right\|^2 \quad k \geq 1. \quad (25)$$

In each simulation, the initial estimates for both PDMM and ADMM were set to be zeros.

We note that ADMM has the free parameter ρ and PDMM has the free parameter γ to be specified. We evaluated the two algorithms for each $\rho = \gamma = 20, 21, 22, \ldots, 110$. For a particular value of ρ (or γ), we counted the number of iterations needed for the algorithm to reach an error below 10^{-3} for the first time.

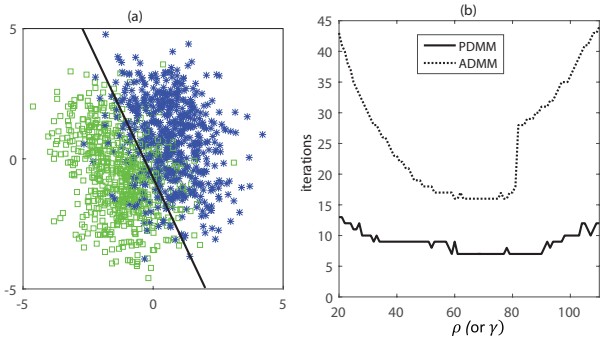

Fig. 1. Experimental comparison of PDMM and ADMM in terms of the convergence speed. In subplot (a), the samples from the two classes are denoted as (blue) $*$ and (green) $\square$, respectively. The line in subplot (a) represents the global solution $(\boldsymbol{w}^{\text{glob}}, b^{\text{glob}})$. In subplot (b), the parameter ρ (or γ) was tested for 20, 21, . . ., 110.

Fig. 1:(b) displays the convergence results of the two algorithms. It is observed that PDMM outperforms ADMM significantly for every value of ρ or γ. Further, the performance of PDMM is less sensitive to the parameter γ than the performance of ADMM to the parameter ρ. This suggests that in practice, the selection of the γ value for PDMM is more flexible than the selection of ρ for ADMM.

The slow convergence of ADMM might be because the algorithm involves the global variable $[\boldsymbol{w}^T, b]^T$. The variable $[\boldsymbol{w}^T, b]^T$ works as a bridge to convey information between the other ones $[\boldsymbol{w}_i^T, b_i]^T$, $i = 1, 2, \ldots, m$. On the other hand, PDMM avoids the global variable $[\boldsymbol{w}^T, b]^T$. As shown in (21), the variable $[\boldsymbol{w}_i^T, b_i]^T$ at node i is able to collect information directly from all other variables $[\boldsymbol{w}_j^T, b_j]^T$, $j \neq i$. As a result, PDMM leads to fast convergence for solving the SVM training problem.

5. CONCLUSION

In this paper, we have firstly revisited PDMM for solving a subclass of the convex problems. By using algebra, we have shown that the updating expressions of PDMM can be simplified considerably, making the algorithm easier to implement in practice. We then apply PDMM to train a SVM over a set of computing units distributively. Experimental results demonstrate that PDMM outperforms ADMM remarkably. Also the experiment suggests that PDMM is less sensitive to the parameter selection than that of ADMM w.r.t. the convergence speed.

6. REFERENCES

[1] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*, Springer, 2015.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *In Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[3] M. Hong, Z. Luo, and M. Razaviyayn, "Convergence Analysis of Alternating Direction Method of Multipliers for a Familty of Nonconvex Problems," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2015.

[4] G. Zhang, R. Heusdens, and W. Bastiaan Kleijn, "On the Convergence Rate of the Bi-Alternating Direction Method of Multipliers," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014, pp. 3897–3901.

[5] G. Zhang and R. Heusdens, "Bi-Alternating Direction Method of Multipliers over Graphs," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2015.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized Gossip Algorithms," *IEEE Trans. Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[7] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[8] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Pattern Recognition*, vol. 2, pp. 121–167, 1998.