Distributed Stochastic Optimization of Regularized Risk via Saddle-Point Problem

Shin Matsushima[1(✉)], Hyokun Yun[2], Xinhua Zhang[3], and S. V. N. Vishwanathan[2,4]

[1] The University of Tokyo, Tokyo, Japan
shin_matsushima@mist.i.u-tokyo.ac.jp
[2] Amazon.com, Seattle, WA 98170, USA
yunhyoku@amazon.com
[3] University of Illinois at Chicago, Chicago, IL 60607, USA
xzhang@uic.edu
[4] University of California, Santa Cruz, CA 95064, USA
vishy@ucsc.edu

Abstract. Many machine learning algorithms minimize a regularized risk, and stochastic optimization is widely used for this task. When working with massive data, it is desirable to perform stochastic optimization in parallel. Unfortunately, many existing stochastic optimization algorithms cannot be parallelized efficiently. In this paper we show that one can rewrite the regularized risk minimization problem as an equivalent saddle-point problem, and propose an efficient distributed stochastic optimization (DSO) algorithm. We prove the algorithm's rate of convergence; remarkably, our analysis shows that the algorithm scales almost linearly with the number of processors. We also verify with empirical evaluations that the proposed algorithm is competitive with other parallel, general purpose stochastic and batch optimization algorithms for regularized risk minimization.

1 Introduction

Regularized risk minimization is a well-known paradigm in machine learning:

$$\min_{\mathbf{w}} P(\mathbf{w}) := \lambda \sum_{j} \phi_j(w_j) + \frac{1}{m}\sum_{i=1}^{m} \ell(\langle \mathbf{w}, \mathbf{x}_i \rangle, y_i). \tag{1}$$

Here, we are given m training data points $\mathbf{x}_i \in \mathbb{R}^d$ and their corresponding labels y_i, while $\mathbf{w} \in \mathbb{R}^d$ is the parameter of the model. Furthermore, w_j denotes the j-th component of $\mathbf{w}$, while $\phi_j(\cdot)$ is a convex function which penalizes complex models. $\ell(\cdot,\cdot)$ is a loss function, which is convex in $\mathbf{w}$. Moreover, $\langle \cdot,\cdot \rangle$ denotes the Euclidean inner product, and $\lambda > 0$ is a scalar which trades-off between the average loss and the regularizer. For brevity, we will use $\ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)$ to denote $\ell(\langle \mathbf{w}, \mathbf{x}_i \rangle, y_i)$.

M. Ceci et al. (Eds.): ECML PKDD 2017, Part I, LNAI 10534, pp. 460–476, 2017.
https://doi.org/10.1007/978-3-319-71249-9_28

Many well-known models can be derived by specializing (1). For instance, if $y_i \in \{\pm 1\}$, then setting $\phi_j(w_j) = \frac{1}{2}w_j^2$ and $\ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) = \max(0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle)$ recovers binary linear support vector machines (SVMs) [23]. On the other hand, using the same regularizer but changing the loss function to $\ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) = \log(1 + \exp(-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle))$ yields regularized logistic regression [11]. Similarly, setting $\phi_j(w_j) = |w_j|$ leads to sparse learning such as LASSO [11] with $\ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) = \frac{1}{2}(y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)^2$.

A number of specialized as well as general purpose algorithms have been proposed for minimizing the regularized risk. For instance, if both the loss and the regularizer are smooth, as is the case with logistic regression, then quasi-Newton algorithms such as L-BFGS [17] have been found to be very successful. On the other hand, for smooth regularizers but non-smooth loss functions, Teo et al. [27] proposed a bundle method for regularized risk minimization (BMRM). Another popular first-order solver is alternating direction method of multipliers (ADMM) [4]. These optimizers belong to the broad class of batch minimization algorithms; that is, in order to perform a parameter update, at every iteration they compute the regularized risk $P(\mathbf{w})$ as well as its gradient

$$\nabla P(\mathbf{w}) = \lambda \sum_{j=1}^{d} \nabla \phi_j(w_j)\, \mathbf{e}_j + \frac{1}{m} \sum_{i=1}^{m} \nabla \ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)\, \mathbf{x}_i, \tag{2}$$

where $\mathbf{e}_j$ denotes the j-th standard basis vector. Both $P(\mathbf{w})$ as well as the gradient $\nabla P(\mathbf{w})$ take $O(md)$ time to compute, which is computationally expensive when m, the number of data points, is large. Batch algorithms can be efficiently parallelized, however, by exploiting the fact that the empirical risk $\frac{1}{m}\sum_{i=1}^{m} \ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)$ as well as its gradient $\frac{1}{m}\sum_{i=1}^{m} \nabla \ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)\, \mathbf{x}_i$ decompose over the data points, and therefore one can compute $P(\mathbf{w})$ and $\nabla P(\mathbf{w})$ in a distributed fashion [7].

Batch algorithms, unfortunately, are known to be unfavorable for large scale machine learning both empirically and theoretically [3]. It is now widely accepted that stochastic algorithms which process one data point at a time are more effective for regularized risk minimization. In a nutshell, the idea here is that (2) can be stochastically approximated by

$$\mathbf{g}_i = \lambda \sum_{j=1}^{d} \nabla \phi_j(w_j)\, \mathbf{e}_j + \nabla \ell_i(\langle \mathbf{w}, \mathbf{x}_i \rangle)\, \mathbf{x}_i, \tag{3}$$

when i is chosen uniformly random in $\{1, \ldots, m\}$. Note that $\mathbf{g}_i$ is an unbiased estimator of the true gradient $\nabla P(\mathbf{w})$; that is, $\mathbb{E}_{i \in \{1,\ldots,m\}}[\mathbf{g}_i] = \nabla P(\mathbf{w})$. Now we can replace the true gradient by this *stochastic* gradient to approximate a gradient descent update as

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}_i, \tag{4}$$

where η is a step size parameter. Computing $\mathbf{g}_i$ only takes $O(d)$ effort, which is independent of m, the number of data points. Bottou and Bousquet [3] show that stochastic optimization is asymptotically faster than gradient descent and other second-order batch methods such as L-BFGS for regularized risk minimization.

However, a drawback of update (4) is that it is not easy to parallelize anymore. Usually, the computation of $\mathbf{g}_i$ in (3) is a very lightweight operation for which parallel speed-up can rarely be expected. On the other hand, one cannot execute multiple updates of (4) simultaneously, since computing $\mathbf{g}_i$ requires *reading* the latest value of $\mathbf{w}$, while updating (4) requires *writing* to the components of $\mathbf{w}$. The problem is even more severe in distributed memory systems, where the cost of communication between processors is significant.

Existing parallel stochastic optimization algorithms try to work around these difficulties in a somewhat ad-hoc manner (see Sect. 4). In this paper, we take a fundamentally different approach and propose a reformulation of the regularized risk (1), for which one can *naturally* derive a parallel stochastic optimization algorithm. Our technical contributions are:

– We reformulate regularized risk minimization as an equivalent saddle-point problem, and show that it can be solved via a new distributed stochastic optimization (DSO) algorithm.
– We prove $O\left(1/\sqrt{T}\right)$ rates of convergence for DSO which is independent of number of processors and theoretically described almost linear dependence of total required time with the number of processors.
– We verify with empirical evaluations that when used for training linear support vector machines (SVMs) or binary logistic regression models, DSO is comparable to general-purpose stochastic (*e.g.*, Zinkevich et al. [33]) or batch (*e.g.*, Teo et al. [27]) optimizers.

2 Reformulating Regularized Risk Minimization

We begin by reformulating the regularized risk minimization problem as an equivalent saddle-point problem. Towards this end, we first rewrite (1) by introducing an auxiliary variable u_i for each $\mathbf{x}_i$:

$$\min_{\mathbf{w},\mathbf{u}} \ \lambda \sum_{j=1}^{d} \phi_j\left(w_j\right) + \frac{1}{m}\sum_{i=1}^{m} \ell_i\left(u_i\right) \quad \text{s.t.} \ \ u_i = \langle \mathbf{w}, \mathbf{x}_i \rangle \quad \forall\, i = 1, \ldots, m. \tag{5}$$

By introducing Lagrange multipliers α_i to eliminate the constraints, we obtain

$$\min_{\mathbf{w},\mathbf{u}} \max_{\boldsymbol{\alpha}} \lambda \sum_{j=1}^{d} \phi_j\left(w_j\right) + \frac{1}{m}\sum_{i=1}^{m} \ell_i\left(u_i\right) + \alpha_i(u_i - \langle \mathbf{w}, \mathbf{x}_i \rangle).$$

Here $\mathbf{u}$ denotes a vector whose components are u_i. Likewise, $\boldsymbol{\alpha}$ is a vector whose components are α_i. Since the objective function (5) is convex and the constraints are linear, strong duality applies [5]. Therefore, we can switch the maximization over $\boldsymbol{\alpha}$ and the minimization over $\mathbf{w}, \mathbf{u}$. Note that $\min_{u_i} \alpha_i u_i + \ell_i(u_i)$ can be written $-\ell_i^\star(-\alpha_i)$, where $\ell_i^\star(\cdot)$ is the Fenchel-Legendre conjugate of $\ell_i(\cdot)$ [5]. The above transformations yield to our formulation:

$$\max_{\boldsymbol{\alpha}} \min_{\mathbf{w}} f\left(\mathbf{w}, \boldsymbol{\alpha}\right) := \lambda \sum_{j=1}^{d} \phi_j\left(w_j\right) - \frac{1}{m} \sum_{i=1}^{m} \alpha_i \left\langle \mathbf{w}, \mathbf{x}_i \right\rangle - \frac{1}{m} \sum_{i=1}^{m} \ell_i^\star\left(-\alpha_i\right). \quad (6)$$

If we analytically minimize $f(\mathbf{w}, \boldsymbol{\alpha})$ in terms of $\mathbf{w}$ to eliminate it, then we obtain so-called *dual* objective which is only a function of $\boldsymbol{\alpha}$. Moreover, any combination of $\mathbf{w}^*$ which is a solution of the primal problem (1), and $\boldsymbol{\alpha}^*$ which is a solution of the dual problem, forms a saddle point of $f\left(\mathbf{w}, \boldsymbol{\alpha}\right)$ [5]. In other words, minimizing the primal, maximizing the dual, and finding a saddle point of $f\left(\mathbf{w}, \boldsymbol{\alpha}\right)$ are all equivalent problems.

2.1 Stochastic Optimization

Let x_{ij} denote the j-th coordinate of $\mathbf{x}_i$, and $\Omega_i := \{j : x_{ij} \neq 0\}$ denote the non-zero coordinates of $\mathbf{x}_i$. Similarly, let $\bar{\Omega}_j := \{i : x_{ij} \neq 0\}$ denote the set of data points where the j-th coordinate is non-zero and $\Omega := \{(i, j) : x_{ij} \neq 0\}$ denotes the set of all non-zero coordinates in the training dataset $\mathbf{x}_1, \ldots, \mathbf{x}_m$. Then, $f(\mathbf{w}, \boldsymbol{\alpha})$ can be rewritten as

$$\begin{aligned} f\left(\mathbf{w}, \boldsymbol{\alpha}\right) &= \sum_{(i,j)\in\Omega} \frac{\lambda \phi_j\left(w_j\right)}{\left|\bar{\Omega}_j\right|} - \frac{\ell_i^\star(-\alpha_i)}{m\left|\Omega_i\right|} - \frac{\alpha_i w_j x_{ij}}{m} \\ &=: \sum_{(i,j)\in\Omega} f_{i,j}\left(w_j, \alpha_i\right), \end{aligned}$$

where $|\cdot|$ denotes the cardinality of a set. Remarkably, each component $f_{i,j}$ in the above summation depends only on one component w_j of $\mathbf{w}$ and one component α_i of $\boldsymbol{\alpha}$. This allows us to derive an optimization algorithm which is stochastic in terms of both i and j. Let us define

$$\mathbf{g}_{i,j} := \left(|\Omega| \left(\frac{\lambda \nabla \phi_j\left(w_j\right)}{\left|\bar{\Omega}_j\right|} - \frac{\alpha_i x_{ij}}{m} \right) \mathbf{e}_j, |\Omega| \left(\frac{\nabla \ell_i^\star(-\alpha_i)}{m\left|\Omega_i\right|} - \frac{w_j x_{ij}}{m} \right) \mathbf{e}_i \right). \quad (7)$$

Under the uniform distribution over $(i, j) \in \Omega$, one can easily see that $\mathbf{g}_{i,j}$ is an unbiased estimate of the gradient of $f\left(\mathbf{w}, \boldsymbol{\alpha}\right)$, that is, $\mathbb{E}_{\{(i,j)\in\Omega\}}\left[\mathbf{g}_{i,j}\right] = \left(\nabla_{\mathbf{w}} f\left(\mathbf{w}, \boldsymbol{\alpha}\right), -\nabla_{\boldsymbol{\alpha}} - f\left(\mathbf{w}, \boldsymbol{\alpha}\right)\right)$. Since we are interested in finding a saddle point of $f\left(\mathbf{w}, \boldsymbol{\alpha}\right)$, our stochastic optimization algorithm uses the stochastic gradient $\mathbf{g}_{i,j}$ to take a *descent* step in $\mathbf{w}$ and an *ascent* step in $\boldsymbol{\alpha}$ [19]:

$$w_j \leftarrow w_j - \eta \left(\frac{\lambda \nabla \phi_j\left(w_j\right)}{\left|\bar{\Omega}_j\right|} - \frac{\alpha_i x_{ij}}{m} \right), \ \alpha_i \leftarrow \alpha_i + \eta \left(\frac{\nabla \ell_i^\star(-\alpha_i)}{m\left|\Omega_i\right|} - \frac{w_j x_{ij}}{m} \right). \quad (8)$$

Surprisingly, the time complexity of update (8) is independent of the size of data; it is $O(1)$. Compare this with the $O(md)$ complexity of batch update and $O(d)$ complexity of regular stochastic gradient descent.

Note that in the above discussion, we implicitly assumed that $\phi_j(\cdot)$ and $\ell_i^{\star}(\cdot)$ are differentiable. If that is not the case, then their derivatives can be replaced by sub-gradients [5]. Therefore this approach can deal with wide range of regularized risk minimization problem.

3 Parallelization

The minimax formulation (6) not only admits an efficient stochastic optimization algorithm, but also allows us to derive a distributed stochastic optimization (DSO) algorithm. The key observation underlying DSO is the following: Given (i, j) and (i', j') both in Ω, if $i \neq i'$ and $j \neq j'$ then one can simultaneously perform update (8) on (w_j, α_i) and $(w_{j'}, \alpha_{i'})$. In other words, the updates to w_j and α_i are independent of the updates to $w_{j'}$ and $\alpha_{i'}$, as long as $i \neq i'$ and $j \neq j'$.

Before we formally describe DSO we would like to present some intuition using Fig. 1. Here we assume that we have 4 processors. The data matrix X is an $m \times d$ matrix formed by stacking $\mathbf{x}_i^{\top}$ for $i = 1, \ldots, m$, while $\mathbf{w}$ and $\boldsymbol{\alpha}$ denote the parameters to be optimized. The non-zero entries of X are marked by an x in the figure. Initially, both parameters as well as rows of the data matrix are partitioned across processors as depicted in Fig. 1 (left); colors in the figure denote ownership *e.g.*, the first processor owns a fraction of the data matrix and a fraction of the parameters $\boldsymbol{\alpha}$ and $\mathbf{w}$ (denoted as $\mathbf{w}^{(1)}$ and $\boldsymbol{\alpha}^{(1)}$) shaded with red. Each processor samples a non-zero entry x_{ij} of X within the dark shaded rectangular region (active area) depicted in the figure, and updates the corresponding w_j and α_i. After performing updates, the processors stop and exchange coordinates of $\mathbf{w}$. This defines an *inner iteration*. After each inner iteration, ownership of the $\mathbf{w}$ variables and hence the active area change, as shown in Fig. 1 (right). If there are p processors, then p inner iterations define an *epoch*. Each coordinate of $\mathbf{w}$ is updated by each processor at least once in an epoch. The algorithm iterates over epochs until convergence.

Four points are worth noting. First, since the active area of each processor does not share either row or column coordinates with the active area of other processors, as per our key observation above, the updates can be carried out by each processor in parallel without any need for intermediate communication with other processors. Second, we partition and distribute the data only once. The coordinates of $\boldsymbol{\alpha}$ are partitioned at the beginning and are not exchanged by the processors; only coordinates of $\mathbf{w}$ are exchanged. This means that the cost of communication is independent of m, the number of data points. Third, our algorithm can work in both shared memory, distributed memory, and hybrid (multiple threads on multiple machines) architectures. Fourth, the $\mathbf{w}$ parameter is distributed across multiple machines and there is no redundant storage, which makes the algorithm scale linearly in terms of space complexity. Compare

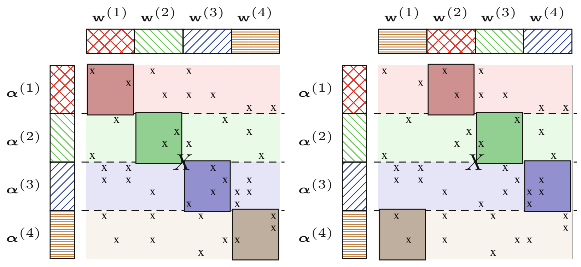
$\mathbf{w}^{(1)}$
$\mathbf{w}^{(2)}$
$\mathbf{w}^{(3)}$
$\mathbf{w}^{(4)}$
$\boldsymbol{\alpha}^{(1)}$
$\boldsymbol{\alpha}^{(2)}$
$\boldsymbol{\alpha}^{(3)}$
$\boldsymbol{\alpha}^{(4)}$
X

Fig. 1. Illustration of DSO with 4 processors. The rows of the data matrix X as well as the parameters $\mathbf{w}$ and $\boldsymbol{\alpha}$ are partitioned as shown. Colors denote ownerships. The active area of each processor is in dark colors. Left: the initial state. Right: the state after one bulk synchronization. (Color figure online)

this with the fact that most parallel optimization algorithms require each local machine to hold a copy of $\mathbf{w}$.

To formally describe DSO, suppose p processors are available, and let $I_1, \ldots, I_p$ denote a fixed partition of the set $\{1, \ldots, m\}$ and $J_1, \ldots, J_p$ denote a fixed partition of the set $\{1, \ldots, d\}$ such that $|I_q| \approx |I_{q'}|$ and $|J_r| \approx |J_{r'}|$ for any $1 \leq q, q', r, r' \leq p$. We partition the data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and the labels $\{y_1, \ldots, y_m\}$ into p disjoint subsets according to $I_1, \ldots, I_p$ and distribute them to p processors. The parameters $\{\alpha_1, \ldots, \alpha_m\}$ are partitioned into p disjoint subsets $\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(p)}$ according to $I_1, \ldots, I_p$ while $\{w_1, \ldots, w_d\}$ are partitioned into p disjoint subsets $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(p)}$ according to $J_1, \ldots, J_p$ and distributed to p processors, respectively. The partitioning of $\{1, \ldots, m\}$ and $\{1, \ldots, d\}$ induces a $p \times p$ partition of Ω:

$$\Omega^{(q,r)} := \{(i,j) \in \Omega \ : \ i \in I_q, j \in J_r\}, \ \ \forall q, r \in \{1, \ldots, p\}.$$

The execution of DSO proceeds in epochs, and each epoch consists of p inner iterations; at the beginning of the r-th inner iteration ($r \geq 1$), processor q owns $\mathbf{w}^{(\sigma_r(q))}$ where $\sigma_r(q) := \{(q + r - 2) \bmod p\} + 1$, and executes stochastic updates (8) on coordinates in $\Omega^{(q,\sigma_r(q))}$. Since these updates only involve variables in $\boldsymbol{\alpha}^{(q)}$ and $\mathbf{w}^{(\sigma(q))}$, no communication between processors is required to execute them. After every processor has finished its updates, $\mathbf{w}^{(\sigma_r(q))}$ is sent to machine $\sigma_{r+1}^{-1}(\sigma_r(q))$ and the algorithm moves on to the $(r+1)$-st inner iteration. Detailed pseudo-code for the DSO algorithm can be found in Algorithm 1.

3.1 Convergence Analysis

It is known that the stochastic procedure in Sect. 2.1 is guaranteed to converge to asaddle point of $f(\mathbf{w}, \boldsymbol{\alpha})$ if (i, j) is randomly accessed [19]. The main technical difficulty in proving convergence in our case is due to the fact that DSO does not sample (i, j) coordinates uniformly at random due to its distributed nature. Therefore, first we prove that DSO is serializable in a certain sense, that is, there exists an ordering of the updates such that *replaying* them on a single machine

Algorithm 1. Distributed stochastic optimization (DSO) for finding saddle point of (6)

```
1: Each processor $q \in \{1, 2, \ldots, p\}$ initializes $\mathbf{w}^{(q)}$, $\boldsymbol{\alpha}^{(q)}$
2: $t \leftarrow 1$
3: repeat
4:    $\eta_t \leftarrow \eta_0 / \sqrt{t}$
5:    for all $r \in \{1, 2, \ldots, p\}$ do
6:       for all processors $q \in \{1, 2, \ldots, p\}$ in parallel do
7:          for $(i, j) \in \Omega^{(q, \sigma_r(q))}$ do
8:             $w_j \leftarrow w_j - \eta_t \left( \frac{\lambda \nabla \phi_j(w_j)}{|\bar{\Omega}_j|} - \frac{\alpha_i x_{ij}}{m} \right)$, $\alpha_i \leftarrow \alpha_i + \eta_t \left( \frac{\nabla \ell_i^\star(-\alpha_i)}{m |\Omega_i|} - \frac{w_j x_{ij}}{m} \right)$
9:          end for
10:         send $\mathbf{w}^{(\sigma_r(q))}$ to machine $\sigma_{r+1}^{-1}(\sigma_r(q))$ and receive $\mathbf{w}^{(\sigma_{r+1}(q))}$
11:      end for
12:   end for
13:   $t \leftarrow t + 1$
14: until convergence
```

would recover the same solution produced by DSO. We then analyze this serial algorithm to establish convergence. We believe that this proof technique is of independent interest, and differs significantly from convergence analysis for other parallel stochastic algorithms which typically assume correlation between data points e.g. [6,15]. We first formally state the main theorem and then prove 3 lemmas. Finally we give a proof of the main theorem in the last part of this section.

Theorem 1. *Let* $(\mathbf{w}^t, \boldsymbol{\alpha}^t)$ *and* $\left(\tilde{\mathbf{w}}^t, \tilde{\boldsymbol{\alpha}}^t\right) := \left(\frac{1}{t}\sum_{s=1}^t \mathbf{w}^s, \frac{1}{t}\sum_{s=1}^t \boldsymbol{\alpha}^s\right)$ *denote the parameter values, and the averaged parameter values respectively after the* t*-th epoch of Algorithm 1. Moreover, assume that* $\|\mathbf{w}\|, \|\boldsymbol{\alpha}\|, |\nabla\phi_j(w_j)|, |\nabla\ell_i^\star(-\alpha_i)|,$ *and* λ *are upper bounded by a constant* $c > 1$*. Then, there exists a constant* C*, which is dependent only on* c*, such that after* T *epochs the duality gap is*

$$\max_{\boldsymbol{\alpha}} f\left(\tilde{\mathbf{w}}^T, \boldsymbol{\alpha}\right) - \min_{\mathbf{w}} f\left(\mathbf{w}, \tilde{\boldsymbol{\alpha}}^T\right) \leq C\frac{\sqrt{d}}{\sqrt{T}}. \tag{9}$$

On the other hand, if $\phi_j(s) = \frac{1}{2}s^2$, $\sqrt{\max_i |\Omega_i|} < m$ *and* $\eta_t < \frac{1}{\lambda}$ *hold, then there exists a different constant* C' *dependent only on* c *and satisfying*

$$\max_{\boldsymbol{\alpha}'} f\left(\tilde{\mathbf{w}}^T, \boldsymbol{\alpha}'\right) - \min_{\mathbf{w}'} f\left(\mathbf{w}', \tilde{\boldsymbol{\alpha}}^T\right) \leq \frac{C'}{\sqrt{T}}. \tag{10}$$

The first lemma states that there exists an ordering of the pairs of coordinates (i, j)s that recovers the solution produced by DSO.

Lemma 1. *On the inner iterations of the* t*-th epoch of Algorithm 1, let us index all* $(i, j) \in \Omega$ *as* (i_k, j_k) *by* $k = 1, \ldots, |\Omega|$ *as follows:* $a < b$ *if updates to* (w_{j_a}, α_{i_a}) *were performed before updating* (w_{j_b}, α_{i_b})*. On the other hand, if* (w_{j_a}, α_{i_a}) *and*

(w_{j_b}, α_{i_b}) were updated at the same time because we have p processors simultaneously updating the parameters, then the updates are ordered according to the rank of the processor performing the update[1]*. Then, denote the parameter values after k updates by $(\mathbf{w}_k^t, \boldsymbol{\alpha}_k^t)$. For all k and t we have*

$$\mathbf{w}_k^t = \mathbf{w}_{k-1}^t - \eta_t \nabla_{\mathbf{w}} f_k \left(\mathbf{w}_{k-1}^t, \boldsymbol{\alpha}_{k-1}^t\right) \tag{11}$$

$$\boldsymbol{\alpha}_k^t = \boldsymbol{\alpha}_{k-1}^t - \eta_t \nabla_{\boldsymbol{\alpha}} - f_k \left(\mathbf{w}_{k-1}^t, \boldsymbol{\alpha}_{k-1}^t\right), \tag{12}$$

where

$$f_k(\mathbf{w}, \boldsymbol{\alpha}) := f_{i_k, j_k}\left(w_{j_k}, \alpha_{i_k}\right).$$

Proof. Let q be the processor which performed the k-th update in the $(t+1)$-st epoch. Moreover, let $(k-\delta)$ be the most recent previous update done by processor q. There exists $1 \leq \delta', \delta'' \leq \delta$ such that $\left(\mathbf{w}_{k-\delta'}^t, \boldsymbol{\alpha}_{k-\delta''}^t\right)$ be the parameter values read by the q-th processor to the perform k-th update. Because of our data partitioning scheme, only q can change the value of the i_k-th component of $\boldsymbol{\alpha}$ and the j_k-th component of $\mathbf{w}$. Therefore, we have

$$\alpha_{k-1,i_k}^t = \alpha_{\kappa,i_k}^t, \text{ and } w_{k-1,j_k} = w_{\kappa,j_k}^t, \quad k-\delta \leq \forall \kappa \leq k-1.$$

Since f_k is invariant to changes in any coordinate other than (i_k, j_k), we have

$$f_k\left(\mathbf{w}_{k-\delta'}^t, \boldsymbol{\alpha}_{k-\delta''}^t\right) = f_k\left(\mathbf{w}_{k-1}^t, \boldsymbol{\alpha}_{k-1}^t\right).$$

The claim holds because we can write the k-th update formula as

$$\mathbf{w}_k^t = \mathbf{w}_{k-1}^t - \eta_t \nabla_{\mathbf{w}} f_k\left(\mathbf{w}_{k-\delta'}^t, \boldsymbol{\alpha}_{k-\delta''}^t\right) \text{ and} \tag{13}$$

$$\boldsymbol{\alpha}_k^t = \boldsymbol{\alpha}_{k-1}^t - \eta_t \nabla_{\boldsymbol{\alpha}} - f_k\left(\mathbf{w}_{k-\delta'}^t, \boldsymbol{\alpha}_{k-\delta''}^t\right). \tag{14}$$

□

Next, we prove the following technical lemma that shows a sufficient condition to establish a global convergence of general iterative algorithms on general convex-concave functions. Note that it is closely related to well-known results on convex functions (e.g., Theorem 3.2.2 in [20], Lemma 14.1. in [24]).

Lemma 2. *Suppose there exists $D > 0$ and $C > 0$ such that for all $(\mathbf{w}, \boldsymbol{\alpha})$ and $(\mathbf{w}', \boldsymbol{\alpha}')$ we have $\|\mathbf{w}-\mathbf{w}'\|^2 + \|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\|^2 \leq D$, and for all $t = 1, \ldots, T$ and all $(\mathbf{w}, \boldsymbol{\alpha})$ we have*

$$\left\|\mathbf{w}^{t+1}-\mathbf{w}\right\|^2 + \left\|\boldsymbol{\alpha}^{t+1}-\boldsymbol{\alpha}\right\|^2 \leq \left\|\mathbf{w}^t-\mathbf{w}\right\|^2 + \left\|\boldsymbol{\alpha}^t-\boldsymbol{\alpha}\right\|^2 - 2\eta_t\left(f\left(\mathbf{w}^t, \boldsymbol{\alpha}\right) - f\left(\mathbf{w}, \boldsymbol{\alpha}^t\right)\right) + C\eta_t^2, \tag{15}$$

then setting $\eta_t = \sqrt{\frac{D}{2Ct}}$ ensures that

$$\max_{\boldsymbol{\alpha}'} f\left(\tilde{\mathbf{w}}^T, \boldsymbol{\alpha}'\right) - \min_{\mathbf{w}'} f\left(\mathbf{w}', \tilde{\boldsymbol{\alpha}}^T\right) \leq \sqrt{\frac{2DC}{T}}. \tag{16}$$

[1] Any other tie-breaking rule would also suffice.

Proof. Rearrange (15) and divide by η_t to obtain

$$2\left(f\left(\mathbf{w}^t,\boldsymbol{\alpha}\right)-f\left(\mathbf{w},\boldsymbol{\alpha}^t\right)\right)\le \eta_t C+\frac{1}{\eta_t}\Big(\left\|\mathbf{w}^t-\mathbf{w}\right\|^2+\left\|\boldsymbol{\alpha}^t-\boldsymbol{\alpha}\right\|^2 \\ -\left\|\mathbf{w}^{t+1}-\mathbf{w}\right\|^2-\left\|\boldsymbol{\alpha}^{t+1}-\boldsymbol{\alpha}\right\|^2\Big).$$

Summing the above for $t=1,\ldots,T$ yields

$$\begin{aligned}2\sum_{t=1}^{T} f\left(\mathbf{w}^t,\boldsymbol{\alpha}\right)-2\sum_{t=1}^{T} f\left(\mathbf{w},\boldsymbol{\alpha}^t\right)&\le\sum_{t=1}^{T}\eta_t C+\frac{1}{\eta_1}\left(\left\|\mathbf{w}^1-\mathbf{w}\right\|^2+\left\|\boldsymbol{\alpha}^1-\boldsymbol{\alpha}\right\|^2\right)\\ &\quad+\sum_{t=2}^{T-1}\left(\frac{1}{\eta_{t+1}}-\frac{1}{\eta_t}\right)\left(\left\|\mathbf{w}^t-\mathbf{w}\right\|^2+\left\|\boldsymbol{\alpha}^t-\boldsymbol{\alpha}\right\|^2\right)\\ &\quad-\frac{1}{\eta_T}\left(\left\|\mathbf{w}^{T+1}-\mathbf{w}\right\|^2+\left\|\boldsymbol{\alpha}^{T+1}-\boldsymbol{\alpha}\right\|^2\right)\\ &\le\sum_{t=1}^{T}\eta_t C+\frac{1}{\eta_1}D+\sum_{t=2}^{T-1}\left(\frac{1}{\eta_{t+1}}-\frac{1}{\eta_t}\right)D\\ &\le\sum_{t=1}^{T}\eta_t C+\frac{1}{\eta_T}D.\end{aligned}\tag{17}$$

On the other hand, thanks to convexity in $\mathbf{w}$ and concavity in $\boldsymbol{\alpha}$, we see

$$f\left(\tilde{\mathbf{w}}^T,\boldsymbol{\alpha}\right)\le\frac{1}{T}\sum_{t=1}^{T} f\left(\mathbf{w}^t,\boldsymbol{\alpha}\right),\quad\text{and}\quad -f\left(\mathbf{w},\tilde{\boldsymbol{\alpha}}^T\right)\le\frac{1}{T}\sum_{t=1}^{T}-f\left(\mathbf{w},\boldsymbol{\alpha}^t\right).$$

Using them for (17) and letting $\eta_t=\sqrt{\frac{D}{2Ct}}$ leads to the following inequalities

$$f\left(\tilde{\mathbf{w}}^T,\boldsymbol{\alpha}\right)-f\left(\mathbf{w},\tilde{\boldsymbol{\alpha}}^T\right)\le\frac{\sum_{t=1}^{T}\eta_t C+\frac{1}{\eta_T}D}{2T}\le\frac{\sqrt{DC}}{2T}\sum_{t=1}^{T}\frac{1}{\sqrt{2t}}+\sqrt{\frac{DC}{2T}}.$$

The claim in (16) follows by using $\sum_{t=1}^{T}\frac{1}{\sqrt{2t}}\le\sqrt{2T}$. □

In order to derive (9), C of (15) has to be the order of d. In case of L_2-regularizer, it has to be dependent only on c to obtain (10). The last lemma validates them. The proof is technical, and related to techniques outlined in Nedić and Bertsekas [18]. See Appendix for the proof.

Lemma 3. *Under the assumptions outlined in Theorem 1, (13) and (14), (15) is satisfied with C of the form of $C=C_1 d$. It does with $C=C_2$ in case of L_2-regularizer. Here C_1 and C_2 are dependent only on c.*

The proof of Theorem 1 can be shown in a very simple form given those 3 lemmas.

Proof. Because the parameter produced by Algorithm 1 is the same as one defined by (13) and (14), it is sufficient to show (13) and (14) lead to the statements in the theorem. From Lemma 3 and the fact that $\|\mathbf{w}-\mathbf{w}'\|^2+\|\boldsymbol{\alpha}-\boldsymbol{\alpha}'\|^2 \leq 8c^2$, (16) of Lemma 2 holds with $\sqrt{CD} = 2c\sqrt{2C_1 d}$ for general case and $\sqrt{CD} = 2c\sqrt{2C_2}$ in case of L_2-regularizer, where C_1 and C_2 are dependent only on c. This immediately implies (9) and (10). $\square$

To understand the implications of the above theorem, let us assume that Algorithm 1 is run with $p \leq \min(m, d)$ processors with a partitioning of Ω such that $\left|\Omega^{(q,\sigma_r(q))}\right| \approx \frac{|\Omega|}{p^2}$ and $|J_q| \approx \frac{d}{p}$ for all q. Let us denote time amount taken in performing updates in one epoch by T_{u}, which is of $\mathcal{O}(|\Omega|)$. Moreover, let us assume that communicating $\mathbf{w}$ across the network takes time amount denoted by T_{c}, which is of $\mathcal{O}(d)$, and communicating a subset of $\mathbf{w}$ takes time proportional to its cardinality.Under these assumptions, the time for each inner iteration of Algorithm 1 can be written as

$$\frac{\left|\Omega^{(q,\sigma_r(q))}\right|}{|\Omega|}T_{\mathrm{u}} + \frac{\left|J_{\sigma_r(q)}\right|}{d}T_{\mathrm{c}} \approx \frac{T_{\mathrm{u}}}{p^2} + \frac{T_{\mathrm{c}}}{p}.$$

Since there are p inner iterations per epoch, the time required to finish an epoch is $T_{\mathrm{u}}/p + T_{\mathrm{c}}$. As per Theorem 1 the number of epochs to obtain an ϵ accurate solution is independent of p. Therefore, one can conclude that DSO scales linearly in p as long as $T_{\mathrm{u}}/T_{\mathrm{c}} \gg p$ holds. As is to be expected, for large enough p the cost of communication T_{c} will eventually dominate.

4 Related Work

Effective parallelization of stochastic optimization for regularized risk minimization has received significant research attention in recent years. Because of space limitations, our review of related work will unfortunately only be partial.

The key difficulty in parallelizing update (4) is that gradient calculation requires us to *read*, while updating the parameter requires us to *write* to the coordinates of $\mathbf{w}$. Consequently, updates have to be executed in serial. Existing work has focused on working around the limitation of stochastic optimization by either (a) introducing strategies for computing the stochastic gradient in parallel (*e.g.*, Langford et al. [15]), (b) updating the parameter in parallel (*e.g.*, Bradley et al. [6], Recht et al. [21]), (c) performing independent updates and combining the resulting parameter vectors (*e.g.*, Zinkevich et al. [33]), or (d) periodically exchanging information between processors (*e.g.*, Bertsekas and Tsitsiklis [2]). While the former two strategies are popular in the shared memory setting, the latter two are popular in the distributed memory setting. In many cases the convergence bounds depend on the amount of correlation between data points and are limited to the case of strongly convex regularizer (Hsieh et al. [12], Yang [30], Zhang and Xiao [32]). In contrast our bounds in Theorem 1 do not depend on such properties of data and more general.

Table 1. Summary of the datasets used in our experiments. m is the total # of examples, d is the # of features, s is the feature density (% of features that are non-zero). K/M/G denotes a thousand/million/billion.

Name	m	d	$\lvert\Omega\rvert$	$s(\%)$
`real-sim`	57.76K	20.95K	2.97M	0.245
`webspam-t`	350.00K	16.61M	1.28G	0.022

Algorithms that use so-called parameter server to synchronize variable updates across processors have recently become popular (*e.g.*, Li et al. [16]). The main drawback of these methods is that it is not easy to "serialize" the updates, that is, to replay the updates on a single machine. This makes proving convergence guarantees, and debugging such frameworks rather difficult, although some recent progress has been made [16].

The observation that updates on individual coordinates of the parameters can be carried out in parallel has been used for other models. In the context of Latent Dirichlet Allocation, Yan et al. [29] used a similar observation to derive an efficient GPU based collapsed Gibbs sampler. On the other hand, for matrix factorization Gemulla et al. [10] and Recht and Ré [22] independently proposed parallel algorithms based on a similar idea. However, to the best of our knowledge, rewriting (1) as a saddle point problem in order to discover parallelism is our novel contribution.

5 Experimental Results

5.1 Dataset and Implementation Details

We implemented DSO, SGD, and PSGD ourselves, while for BMRM we used the optimized implementation that is available from the toolkit for advanced optimization (TAO, https://bitbucket.org/sarich/tao-2.2). All algorithms are implemented in `C++` and use `MPI` for communication. In our multi-machine experiments, each algorithm was run on four machines with eight cores per machine. DSO, SGD, and PSGD used AdaGrad [8] step size adaptation. We also used stochastic variance reduced gradient (SVRG) of Johnson and Zhang [14] to accelerate updates of DSO. In the multi-machine setting DSO initializes parameters of each MPI process by locally executing twenty iterations of dual coordinate descent [9] on its local data to locally initialize w_j and α_i parameters; then w_j values were averaged across machines. We chose binary logistic regression and SVM as test problems, i.e., $\phi_j(s) = \frac{1}{2}s^2$ and $\ell_i(u) = \log(1 + \exp(-u)), [1 - u]_+$. To prevent degeneracy in logistic regression, values of α_i's are restricted to $(10^{-14}, 1 - 10^{-14})$, while in the case of linear SVM they are restricted to $[0, 1]$. Similarly, the w_j's are restricted to lie in the interval $[-1/\sqrt{\lambda}, 1/\sqrt{\lambda}]$ for linear SVM and $[-\sqrt{\log(2)/\lambda}, \sqrt{\log(2)/\lambda}]$ for logistic regression, following the idea of Shalev-Shwartz et al. [25].

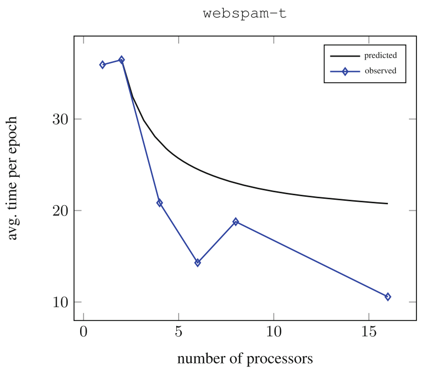

Fig. 2. The average time per epoch using p machines on the `webspam-t` dataset.

5.2 Scalability of DSO

We first verify, that the per epoch complexity of DSO scales as $T_{\mathrm{u}}/p + T_{\mathrm{c}}$, as predicted by our analysis in Sect. 3.1. Towards this end, we took the `webspam-t` dataset of Webb et al. [28], which is one of the largest datasets we could comfortably fit on a single machine. We let $p = \{1, 2, 4, 8, 16\}$ while fixing the number of cores on each machine to be 4.

Using the average time per epoch on one and two machines, one can estimate T_{u} and T_{c}. Given these values, one can then predict the time per iteration for other values of p. Figure 2 shows the predicted time and the measured time averaged over 40 epochs. As can be seen, the time per epoch indeed goes down as $\approx 1/p$ as predicted by the theory. The test error and objective function values on multiple machines was very close to the test error and objective function values observed on a single machine, thus confirming Theorem 1.

5.3 Comparison with Other Solvers

In our single machine experiments we compare DSO with stochastic gradient descent (SGD) and bundle methods for regularized risk minimization (BMRM) of Teo et al. [27]. In our multi-machine experiments we compare with parallel stochastic gradient descent (PSGD) of Zinkevich et al. [33] and BMRM. We chose these competitors because, just like DSO, they are general purpose solvers for regularized risk minimization (1), and hence can solve non-smooth problems such as SVMs as well as smooth problems such as logistic regression. Moreover, BMRM is a specialized solver for regularized risk minimization, which has similar performance to other first-order solvers such as ADMM.

We selected two representative datasets and two values of the regularization parameter $\lambda = \left\{10^{-5}, 10^{-6}\right\}$ to present our results. For the single machine experiments we used the `real-sim` dataset from Hsieh et al. [13], while for the multi-machine experiments we used `webspam-t`. Details of the datasets can be

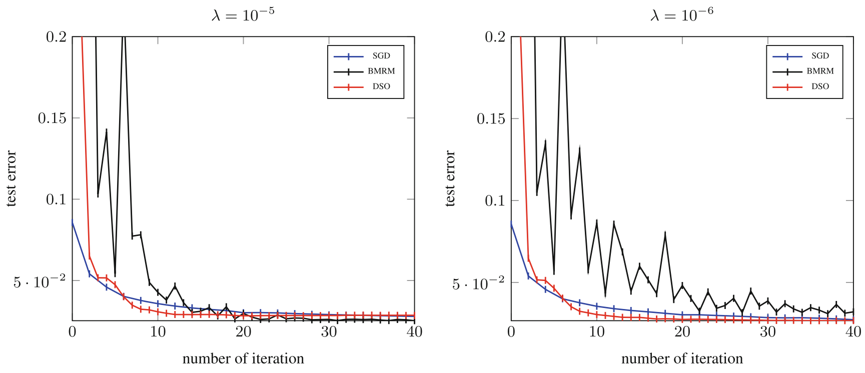

Fig. 3. The test error of different optimization algorithms on linear SVM with `real-sim` dataset, as a function of the number of iteration.

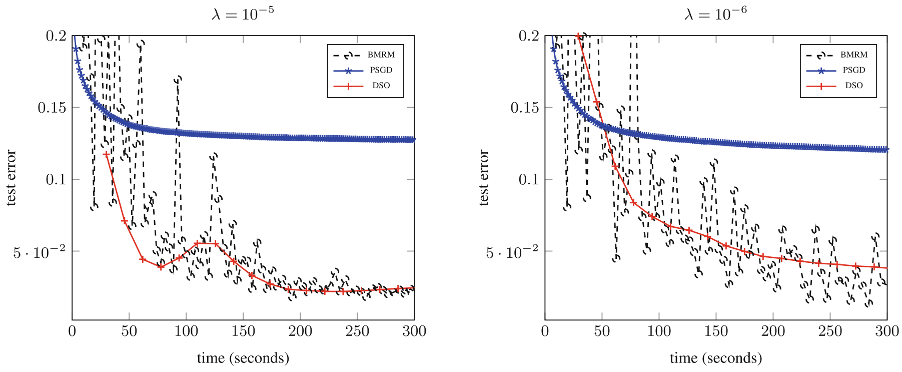

Fig. 4. The test error of different optimization algorithms on logistic regression with `webspam-t` dataset. Test error as a function of elapsed time.

found in Table 1 in the appendix. We use test error rate as comparison metric, since stochastic optimization algorithms are efficient in terms of minimizing generalization error, not training error [3]. The results for single machine experiments on linear SVM training can be found in Fig. 3. As can be seen, DSO shows comparable efficiency to that of SGD, and outperforms BMRM. This demonstrates that saddle-point optimization is a viable strategy even in serial setting.

Our multi-machine experimental results for linear SVM training can be found in Fig. 5. As can be seen, PSGD converges very quickly, but the quality of the final solution is poor; this is probably because PSGD only solves processor-local problems and does not have a guarantee to converge to the global optimum. On the other hand, both BMRM and DSO converges to similar quality solutions, and at fairly comparable rates. Similar trends we observed on logistic regression. Therefore we only show the results with 10^{-5} in Fig. 4.

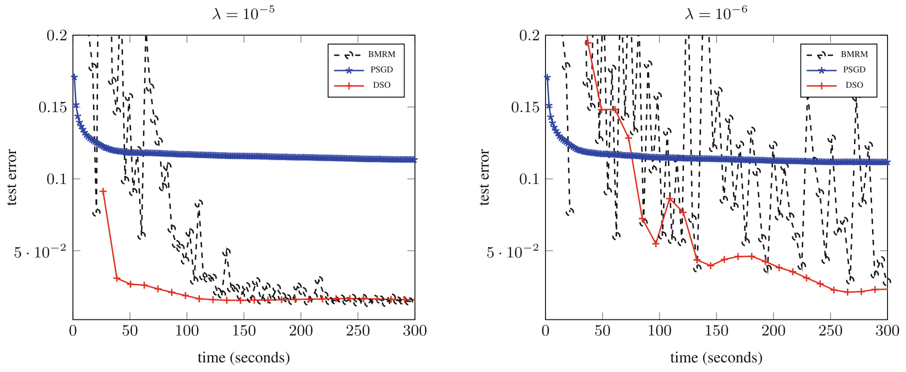

Fig. 5. Test errors of different parallel optimization algorithms on linear SVM with `webspam-t` dataset, as a function of elapsed time.

5.4 Terascale Learning with DSO

Next, we demonstrate the scalability of DSO on one of the largest publicly available datasets. Following the same experimental setup as Agarwal et al. [1], we work with the splice site recognition dataset [26] which contains 50 million training data points, each of which has around 11.7 million dimensions. Each datapoint has approximately 2000 non-zero coordinates and the entire dataset requires around **3 TB** of storage. Previously [26], it has been shown that subsampling reduces performance, and therefore we need to use the entire dataset for training.

Similar to Agarwal et al. [1], our goal is not to show the best classification accuracy on this data (this is best left to domain experts and feature designers). Instead, we wish to demonstrate the scalability of DSO and establish that (a) it can scale to such massive datasets, and (b) the empirical performance as measured by AUPRC (Area Under Precision-Recall Curve) improves as a function of time.

We used 14 machines with 8 cores per machine to train a linear SVM, and plot AUPRC as a function of time in Fig. 6. Since PSGD did not perform well in earlier experiments, here we restrict our comparison to BMRM. This experiment demonstrates one of the advantages of stochastic optimization, namely that the test performance increases steadily as a function of the number of iterations. On the other hand, for a batch solver like BMRM the AUPRC fluctuates as a function of the iteration number. The practical consequence of this observation is that, one usually needs to wait for a batch optimizer to converge before using the resulting solution. On the other hand, even the partial solutions produced by a stochastic optimizer such as DSO usually exhibit good generalization properties.

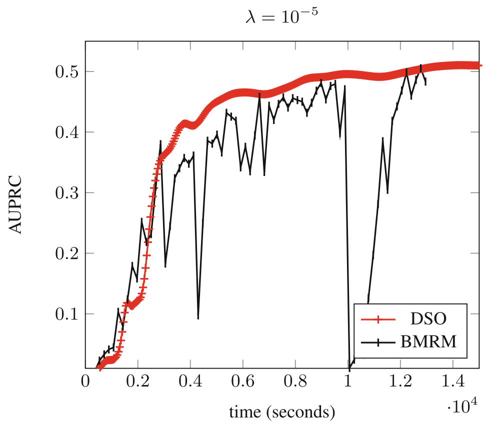

Fig. 6. AUPRC (Area Under Precision-Recall Curve) as a function of elapsed time on linear SVM with splice site recognition dataset.

6 Discussion and Conclusion

We presented a new reformulation of regularized risk minimization as a saddle point problem and showed that one can derive an efficient distributed stochastic optimizer (DSO). We also proved rates of convergence of DSO. Unlike other solvers, our algorithm does not require strong convexity and thus has wider applicability. Our experimental results show that DSO is competitive with state-of-the-art optimizers such as BMRM and SGD, and outperforms simple parallel stochastic optimization algorithms such as PSGD.

A natural next step is to derive an asynchronous version of DSO algorithm along the lines of the NOMAD algorithm proposed by Yun et al. [31]. We can see that our convergence proof which only relies on having an equivalent serial sequence of updates will still apply. Of course, there is also more room to further improve the performance of DSO by deriving better step size adaptation schedules, and exploiting memory caching to speed up random access.

Acknowledgments. This work is partially supported by MEXT KAKENHI Grant Number 26730114 and JST-CREST JPMJCR1304.

References

1. Agarwal, A., Chapelle, O., Dudík, M., Langford, J.: A reliable effective terascale linear learning system. JMLR **15**, 1111–1133 (2014)
2. Bertsekas, D., Tsitsiklis, J.: Parallel and Distributed Computation: Numerical Methods (1997)
3. Bottou, L., Bousquet, O.: The tradeoffs of large-scale learning. In: Optimization for Machine Learning (2011)

4. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends ML **3**(1), 1–123 (2010)
5. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
6. Bradley, J., Kyrola, A., Bickson, D., Guestrin, C.: Parallel coordinate descent for L1-regularized loss minimization. In: ICML, pp. 321–328 (2011)
7. Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G., Ng, A.Y., Olukotun, K.: Map-reduce for machine learning on multicore. In: NIPS, pp. 281–288 (2006)
8. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. JMLR **12**, 2121–2159 (2010)
9. Fan, R.E., Chang, J.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: a library for large linear classification. JMLR **9**, 1871–1874 (2008)
10. Gemulla, R., Nijkamp, E., Haas, P.J., Sismanis, Y.: Large-scale matrix factorization with distributed stochastic gradient descent. In: KDD, pp. 69–77 (2011)
11. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. (2009)
12. Hsieh, C.J., Yu, H.F., Dhillon, I.S.: PASSCoDe: parallel asynchronous stochastic dual coordinate descent. In: ICML (2015)
13. Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: ICML, pp. 408–415 (2008)
14. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: NIPS, pp. 315–323 (2013)
15. Langford, J., Smola, A.J., Zinkevich, M.: Slow learners are fast. In: NIPS (2009)
16. Li, M., Andersen, D.G., Smola, A.J., Yu, K.: Communication efficient distributed machine learning with the parameter server. In: Neural Information Processing Systems (2014)
17. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**(3), 503–528 (1989)
18. Nedić, A., Bertsekas, D.P.: Incremental subgradient methods for nondifferentiable optimization. SIAM J. Optim. **12**(1), 109–138 (2001)
19. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM J. Optim. **19**(4), 1574–1609 (2009)
20. Nesterov, Y.: Introductory Lectures on Convex Optimization: A Basic Course. Springer, Heidelberg (2004). https://doi.org/10.1007/978-1-4419-8853-9
21. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: a lock-free approach to parallelizing stochastic gradient descent. In: NIPS, pp. 693–701 (2011)
22. Recht, B., Ré, C.: Parallel stochastic gradient algorithms for large-scale matrix completion. Math. Program. Comput. **5**(2), 201–226 (2013)
23. Schölkopf, B., Smola, A.J.: Learning with Kernels (2002)
24. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning. Cambridge University Press, Cambridge (2014)
25. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for SVM. In: ICML (2007)
26. Sonnenburg, S., Franc, V.: COFFIN: a computational framework for linear SVMs. In: ICML (2010)
27. Teo, C.H., Vishwanthan, S.V.N., Smola, A.J., Le, Q.V.: Bundle methods for regularized risk minimization. JMLR **11**, 311–365 (2010)
28. Webb, S., Caverlee, J., Pu, C.: Introducing the webb spam corpus: using email spam to identify web spam automatically. In: CEAS (2006)

29. Yan, F., Xu, N., Qi, Y.: Parallel inference for latent Dirichlet allocation on graphics processing units. In: NIPS, pp. 2134–2142 (2009)
30. Yang, T.: Trading computation for communication: distributed stochastic dual coordinate ascent. In: NIPS (2013)
31. Yun, H., Yu, H.F., Hsieh, C.J., Vishwanathan, S.V.N., Dhillon, I.S.: NOMAD: non-locking, stOchastic multi-machine algorithm for asynchronous and decentralized matrix completion. VLDB **7**, 975–986 (2014)
32. Zhang, Y., Xiao, L.: DiSCO: distributed optimization for self-concordant empirical loss. In: ICML (2015)
33. Zinkevich, M., Smola, A.J., Weimer, M., Li, L.: Parallelized stochastic gradient descent. In: NIPS, pp. 2595–2603 (2010)