The AAAI-17 Workshop on
Distributed Machine Learning
WS-17-08

Distributed Inexact Damped Newton Method: Data Partitioning and Work-Balancing

Chenxin Ma, Martin Takáč
Industrial and Systems Engineering Department, Lehigh University
H.S. Mohler Laboratory, 200 West Packer Avenue
Bethlehem, PA 18015

Abstract

In this paper, we study inexact damped Newton method implemented in a distributed environment. We are motivated by the original DiSCO algorithm [Communication-Efficient Distributed Optimization of Self-Concordant Empirical Loss, Yuchen Zhang and Lin Xiao, 2015]. We show that this algorithm may not scale well and propose algorithmic modifications which lead to fewer communications and better load-balancing between nodes. Those modifications lead to a more efficient algorithm with better scaling. This was made possibly by introducing our new pre-conditioner which is specially designed so that the preconditioning step can be solved exactly and efficiently. Numerical experiments for minimization of regularized empirical loss with a 273GB instance shows the efficiency of proposed algorithm.

Introduction

As the size of the datasets becomes larger and larger, distributed optimization methods for machine learning have become increasingly important (Dekel et al. 2012; Shamir and Srebro 2014). Existing methods often require a significant amount of communication between computing nodes (Ma et al. 2015; Yang et al. 2013), which is typically several magnitudes slower than reading data from their memory (Marecek, Richtárik, and Takáč 2014). Thus, distributed machine learning suffers from the communication bottleneck in real world applications.

In this paper we focus on the regularized empirical risk minimization problem

$$\min_{w\in\mathbb{R}^d} f(w) := \tfrac{1}{n}\textstyle\sum_{i=1}^n \phi_i(w,x_i) + \tfrac{\lambda}{2}\|w\|_2^2, \qquad \text{(P)}$$

where $\{x_i,y_i\}_{i=1}^n$ are our training samples with $(x_i,y_i)\in\mathbb{R}^d\times\mathbb{R}$. We will denote by X the data matrix, i.e. $X := [x_1,...,x_n] \in \mathbb{R}^{d\times n}$. We assume that each $\phi_i(\cdot,x_i)$ is L-smooth[1] loss function which typically depends on y_i. The second part of objective function (P) is ℓ_2 regularizer ($\lambda > 0$) which helps to prevent over-fitting of the data. There has been an enormous interest in large-scale machine learning problems and many parallel (Bradley et al. 2011; Recht et al. 2011) or distributed (Agarwal and Duchi 2011; Richtárik and Takáč 2013; Shamir, Srebro, and Zhang 2013) algorithms have been proposed.

From optimization point of view some researches try to minimize (P) directly, which includes SGD (Shalev-Shwartz et al. 2011), SVRG and S2GD (Johnson and Zhang 2013; Konečný et al. 2014) and SAG/SAGA (Schmidt, Roux, and Bach 2013; Defazio, Bach, and Lacoste-Julien 2014). On the other side, some researchers prefer to optimize its dual problem (Hsieh et al. 2008)

$$\max_{\alpha\in\mathbb{R}^n} D(\alpha) := -\tfrac{1}{n}\textstyle\sum_{i=1}^n \phi_i^*(-\alpha_i) - \tfrac{\lambda}{2}\|\tfrac{1}{\lambda n}X\alpha\|^2, \quad \text{(D)}$$

where ϕ_i^* is a convex conjugate function of ϕ_i. This has been done successfully in multicore or distributed settings (Takáč et al. 2013; Jaggi et al. 2014; Ma et al. 2015; Takáč, Richtárik, and Srebro 2015; Qu, Richtárik, and Zhang 2015).

The Challenge In Distributed Computing. We can identify few challenges when we deal with high-performance distributed environment.

1. **Load-Balancing.** Assume that we have m computational nodes available for use. In order to have an algorithm, which is scalable, the algorithm should make each node "equally" busy. Amdahl's law (Rodgers 1985) implies that if the parallel/distributed algorithm runs e.g. 50% of the time only on one of the nodes (usually the master node), then the possible speed-up (on the nodes used) of the algorithm is bounded by $\frac{1}{0.5+0.5/m} \overset{m\to\infty}{\longrightarrow} 2$. Hence, any algorithm which is targeted for a very large scale problems has to be designed in such a way, that the sequential portion of the algorithm is negligible.
2. **Communication efficiency.** As it was stressed in the introduction, in a distributed setting the communication between nodes should be avoided or minimized (if possible). Hence, another –and critical– challenge is to balance the time the nodes are doing some computation and the time they spent in the communication.

In this paper, we modify the design of promising DiSCO algorithm (Zhang and Xiao 2015). We completely redesign the algorithm (partitioning of the data, preconditioning, communication patterns) to get a new algorithm which

1. has **almost linear scaling** – the serial portion of the proposed algorithm is almost negligible,

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]Function ϕ is L-smooth, if $\nabla\phi(\cdot)$ is L-Lipschitz continuous.

2. **balances work-load across nodes perfectly** – all nodes are working all the time, no special job for master node,
3. **small amount of communication** – our algorithm –in some regimes– send a smaller amount of data over the network than in the original DiSCO algorithm.

Related Work

As stressed in the previous section, one of the main bottlenecks in distributed computing is communication. This challenge was handled by many researchers differently. In the ideal case, one would like never to communicate or maybe communicate only once at the end to form a result. However, such a procedure which could communicate only max once and would be able to achieve an arbitrary good solution (when no node can have access to all the data) is more fantasy than reality.

Hence, to somehow "synchronize" work on different computing nodes, researchers use a various standard technique from optimization. Few of them based their algorithms on ADMM type methods (Boyd et al. 2011; Deng and Yin 2012), another used block-coordinate type algorithms (Yang 2013; Jaggi et al. 2014; Ma et al. 2015), where they solved on each node some local sub-problems which together formed an upper-bound on the optimization problem. The balancing of computation and communication was achieved by varying the accuracy of the solutions of the local sub-problems, which turns out to be more efficient than some previous approaches (Takáč et al. 2013; Takáč, Richtárik, and Srebro 2015)). The very recent DANE (Shamir, Srebro, and Zhang 2013) and DiSCO (Zhang and Lin 2015) algorithm are Newton-type methods, where in each iteration the step is solved inexactly using Preconditioned Conjugate Gradient (PCG) method.

Contributions

In this section, we summarize the main contributions of this paper (not in order of significance).

1. **Preconditioning is solved in closed form and efficiently.** The PCG method needs to solve the preconditioned system of linear equations (Step 7 in Algorithm 2). In DiSCO algorithm, the authors suggested using SAG/SAGA algorithm which has a linear rate of convergence. However, such algorithm is run only on the master node, *while all other workers are being idle,* and unfortunately, the *time to solve the preconditioned system is not at all negligible.* In our experiments, we observed that for same dataset the percentage of time spent in solving PCG was more than 50% of the total running time. This implies that DiSCO algorithm would scale very poorly.

   Another problem is, even though it takes such a long time to solve the preconditioning step, exact solutions are still not obtained. Therefore, original DiSCO algorithm uses *approximated PCG* in which in each iteration different matrix P is used (hence it loses the nice property of PCG), which may lead to slow convergence of PCG in practice. Let us note that the theory in (Zhang and Lin 2015) assumes that PCG step is solved exactly.

   To overcome that issue, we propose a new preconditioning matrix P, which can be viewed as an approximated or stochastic Hessian. By exploring the structure of the new preconditioning matrix P, the linear system can be solved much more efficiently. Because, the matrix P is constructed only based on $\tau \ll n$ samples, the time needed to solve the preconditioning system is negligible. Moreover, by applying Woodbury Formula to solve such a linear system with the new P, the solution is guaranteed to be exact, which fixes the problem on inexact solutions in original DiSCO. By applying these approaches, we proposed a variant of DiSCO algorithm called DiSCO-S. Our practical experiments not only confirms that this preconditioning is superior to the preconditioning suggested in original DiSCO algorithm, but also demonstrate that a small τ would give a good performance.
2. **Data Partitioned by Features.** In our setting, we assume that the dataset is large enough that it cannot be stored entirely on any single node and hence the dataset has to be partitioned.

   Both DiSCO and DiSCO-S algorithms are based on partitioning dataset by samples. By considering another way of making partitions, i.e., partitioning by features, we proposed a new DiSCO-F algorithm. In this new setting, the number of communications is reduced by half compared to the original DiSCO. In the DiSCO-F algorithm, all machines will do the same work, and the computation will be distributed more properly. Hence it can be possible to obtain almost linear speed-up.

Assumptions

We assume that the loss function ϕ_i is convex and self-concordant (Zhang and Xiao 2015):

Assumption 1. *For all $i \in [n] := \{1, 2, \dots, n\}$ the convex function ϕ is self-concordant with parameter M i.e. the following inequality holds:*

$$|u^T(f'''(w)[u])u| \le M(u^T f''(w)u)^{\frac{3}{2}} \tag{1}$$

for any $u \in \mathbb{R}^d$ and $w \in dom(f)$, where $f'''(w)[u] := \lim_{t\to 0} \frac{1}{t}(f''(w+tu) - f''(w))$.

Table 1 lists some examples of loss functions which satisfy the Assumption 1 with corresponding constant M.

Table 1: Loss functions satisfying Assumption 1 and the parameter M.

	$\phi_i(w, x_i)$	M
quadratic loss	$(y_i - w^T x_i)^2$	0
squared hinge loss	$(\max\{0, y_i - w^T x_i\})^2$	0
logistic loss	$\log(1 + \exp(-y_i w^T x_i))$	1

We further assume that the function f is both L-smooth and λ-strongly convex.

Assumption 2. *The function $f : \mathbb{R}^d \to \mathbb{R}$ is trice continuously differentiable, and there exist constants $L \ge \lambda > 0$ such that $\forall w \in \mathbb{R}^d$ $\lambda I \preceq f''(w) \preceq LI$, where $f''(w)$ denotes the Hessian of f at w, and I is the $d \times d$ identity matrix.*

Algorithm 1 High-level DiSCO algorithm

1: **Input:** parameters $\rho, \mu \geq 0$, number of iterations K
2: Initializing w_0.
3: **for** k = 0,1,2,...,K **do**
4: Option 1: Given w_k, run DiSCO-S PCG Algorithm 2, get v_k and δ_k
5: Option 2: Given w_k, run DiSCO-F PCG Algorithm 3, get v_k and δ_k
6: Update $w_{k+1} = w_k - \frac{1}{1+\delta_k} v_k$
7: **end for**
8: **Output:** w_{K+1}

The Algorithm

We assume that we have m machines (computing nodes) available which can communicate with each other over the network. We assume that space needed to store the data matrix X exceeds the memory of every single node. Thus we have to split the data (matrix X) over the m nodes. The natural question is: *How to split the data into m parts?* There are many possible ways, but two obvious ones:

1. split the data matrix X by rows (i.e. create m blocks by rows); Because rows of X corresponds to features, we denote the algorithm which is using this type of partitioning as *DiSCO-F*;
2. split the data matrix X by columns; Let us note that columns of X corresponds to samples we denote the algorithm which is using this type of partitioning as *DiSCO-S*;

Notice that the DiSCO-S is the same as DiSCO proposed and analyzed in (Zhang and Xiao 2015). In each iteration of Algorithm 1, wee need to compute an inexact Newton step v_k such that $\|f''(w_k)v_k - \nabla f'(w_k)\|_2 \leq \epsilon_k$, which is an approximate solution to the Newton system $f''(w_k)v_k = \nabla f(w_k)$. The discussion about how to choose ϵ_k and K and a convergence guarantees for Algorithm 1 can be found in (Zhang and Xiao 2015). And the main convergence result still applies here: If Algorithm 2 or 3 is run starting with w^0 then after

$$T \sim \mathcal{O}\Big((f(w^0) - f(w^*) + \log(1/\epsilon)) \sqrt{1 + 2\mu/\lambda} \Big)$$

communication rounds (iterations) the algorithm will produce a solution $\hat{w}$ satisfying $f(\hat{w}) - f(w^*) < \epsilon$, where μ is defined later after (3).

The primary goal of this work is to analyze the algorithmic modifications to DiSCO-S when the partitioning type is changed. It turns out that partitioning on features (DiSCO-F) can lead to an algorithm which uses fewer communications (depending on the relations between d and n, see Table 2).

DiSCO-S Algorithm. If the dataset is partitioned by samples, such that j–th node will only store $X_j = [x_{j,1}, ..., x_{j,n_j}] \in \mathbb{R}^{d \times n_j}$, which is a part of X, then each machine can evaluate a local empirical loss function

$$f_j(w) := \tfrac{1}{n_j} \textstyle\sum_{i=1}^{n_j} \phi(w, x_{j,i}) + \tfrac{\lambda}{2}\|w\|_2^2. \quad (2)$$

Because $\{X_j\}$ is a partition of X we have $\sum_{j=1}^m n_j = n$, our goal now becomes to minimize the function $f(w) = \frac{1}{m}\sum_{h=1}^m f_j(w)$. Let H denote the Hessian $f''(w_k)$. For simplicity in this section we present it only for square loss (and hence in this case $f''(w_k)$ is constant – independent on w_k). However, it naturally extends to any smooth loss.

Algorithm 2 Distributed DiSCO-S: PCG algorithm – data partitioned by samples

1: **Input:** $w_k \in \mathbb{R}^d$, and $\mu \geq 0$. Compute $\nabla f_i(w_k)$ across all nodes.
2: **Initialization:** Let P be computed as (3). $v_0 = 0$, $s_0 = P^{-1}r_0$, $r_0 = \nabla f(w_k)$, $u_0 = s_0$.
3: **for** $t = 0, 1, 2, ...$ **do**
4: Compute Hu_t using all nodes
5: Compute $\alpha_t = \frac{\langle r_t, s_t \rangle}{\langle u_t, Hu_t \rangle}$
6: Update $v_{(t+1)} = v_t + \alpha_t u_t$, $Hv_{(t+1)} = Hv_t + \alpha_t Hu_t$, $r_{t+1} = r_t - \alpha_t Hu_t$.
7: Update $Ps_{(t+1)} = r_{(t+1)}$.
8: Compute $\beta_t = \frac{\langle r_{(t+1)}, s_{(t+1)} \rangle}{\langle r_t, s_t \rangle}$
9: Update $u_{(t+1)} = s_{(t+1)} + \beta_t u_t$.
10: **until:** $\|r_{(r+1)}\|_2 \leq \epsilon_k$
11: **end for**
12: **Return:** $v_k = v_{t+1}$, $\delta_k = \sqrt{v_{(t+1)}^T H v_t + \alpha_t v_{(t+1)}^T H u_t}$

In Algorithm 2, each machine will use its local data to compute the local gradient and local Hessian and then aggregate them together. We also have to choose one machine as the master, which computes all the vector operations of PCG loops (Preconditioned Conjugate Gradient), i.e., step 5-9 in Algorithm 2.

The preconditioning matrix for PCG is defined only on master node and consists of the local Hessian approximated by a subset of data available on master node with size τ, i.e.

$$P = \tfrac{1}{\tau} \textstyle\sum_{j=1}^{\tau} \phi''(w, x_{1,j}) + (\lambda + \mu)I. \quad (3)$$

Here μ is a small regularization parameter satisfying $\|\frac{1}{\tau}\sum_{j=1}^{\tau}\phi''(w, x_{1,j}) + \lambda I - H\| \leq \mu$ (Zhang and Xiao 2015), where H is the true Hessian of objective function $f(\cdot)$. Algorithm 2 presents the distributed PCG mathod for solving the linear system

$$Hv_k = \nabla f(w_k). \quad (4)$$

Notice that in Algorithm 2, there is another linear system $s = P^{-1}r$ to be solved, which has the same dimension as (4). However, because we only apply a subset of data to compute the preconditioning matrix P, this can be solved by Woodbury formula (Press et al. 2007), which will be described detail in the next section.

DiSCO-F Algorithm. If the dataset is partitioned by features, then j-th machine will store $X_j = [a_1^{[j]}, ..., a_n^{[j]}] \in \mathbb{R}^{d_j \times n}$, which contains all the samples, but only with a subset of features. In this case, each machine will only store $w_k^{[j]} \in \mathbb{R}^{d_j}$ and thus is responsible for the computation and updates of $\mathbb{R}^{d_j}$ vectors only. By doing so, we only need one communicate (using *MPI_Allreduce*) on a vector of length n, in addition to two *MPI_Allreduce* on scalars numbers.

Algorithm 3 Distributed DiSCO-F: PCG algorithm – data partitioned by features

1: **Input:** $w_k^{[j]} \in \mathbb{R}^{d_j}$ for $j = 1, 2, ..., m$, and $\mu \geq 0$.
2: **Initialization:** Let P be computed as (3). $v_0^{[j]} = 0$, $s_0^{[j]} = (P^{-1})^{[j]} r_0^{[j]}$, $r_0^{[j]} = f'(w_k^{[j]})$, $u_0^{[j]} = s_0^{[j]}$.
3: **while** $\|r_{r+1}\|_2 \leq \epsilon_k$ **do**
4: Compute $(Hu_t)^{[j]}$ by communication
5: Compute $\alpha_t = \frac{\sum_{j=1}^m \langle r_t^{[j]}, s_t^{[j]} \rangle}{\sum_{j=1}^m \langle u_t^{[j]}, (Hu_t)^{[j]} \rangle}$ by communication
6: Update $v_{t+1}^{[j]} = v_t^{[j]} + \alpha_t u_t^{[j]}$, $(Hv_{t+1})^{[j]} = (Hv_t)^{[j]} + \alpha_t (Hu_t)^{[j]}$, $r_{t+1}^{[j]} = r_t^{[j]} - \alpha_t (Hu_t)^{[j]}$.
7: Update $P^{[j]} s_{t+1}^{[j]} = r_{t+1}^{[j]}$.
8: Compute $\beta_t = \frac{\sum_{j=1}^m \langle r_{t+1}^{[j]}, s_{t+1}^{[j]} \rangle}{\sum_{j=1}^m \langle r_t^{[j]}, s_t^{[j]} \rangle}$ by communication
9: Update $u_{t+1}^{[j]} = s_{t+1}^{[j]} + \beta_t u_t^{[j]}$.
10: **end while**
11: Compute $\delta_k^{[j]} = \sqrt{{v_{t+1}^{[j]}}^T (Hv_t)^{[j]} + \alpha_t {v_{t+1}^{[j]}}^T (Hu_t)^{[j]}}$.
12: **Integration:** $v_k = [v_{t+1}^{[1]}, ..., v_{t+1}^{[m]}]$, $\delta_k = [\delta_{t+1}^{[1]}, ..., \delta_{t+1}^{[m]}]$
13: **Return:** v_k, δ_k

Table 2: Comparison of computation between different algorithms. For DiSCO-S, there exist large differences of computation between the master node and another node. However, for DiSCO-F, each node does the same amount of computation.

	Operation	DiSCO-S	DiSCO-F
master	$y = Mx$	$1(\mathbb{R}^{d\times d} \cdot \mathbb{R}^d)$	$1(\mathbb{R}^{d_1\times d_1} \cdot \mathbb{R}^{d_1})$
	$Mx = y$	$1\ (\mathbb{R}^d)$	$1\ (\mathbb{R}^{d_1})$
	$x + y$	$4\ (\mathbb{R}^d)$	$4\ (\mathbb{R}^{d_1})$
	$x^T y$	$4\ (\mathbb{R}^d)$	$4\ (\mathbb{R}^{d_1})$
nodes	$y = Mx$	$1\ (\mathbb{R}^{d\times d} \cdot \mathbb{R}^d)$	$1(\mathbb{R}^{d_1\times d_i} \cdot \mathbb{R}^{d_i})$
	$Mx = y$	0	$1\ (\mathbb{R}^{d_i})$
	$x + y$	0	$4\ (\mathbb{R}^{d_i})$
	$x^T y$	0	$4\ (\mathbb{R}^{d_i})$

Comparison of a Communication and a Computational Cost. In Table 2 we compare the communication cost for the two approaches DiSCO-S/DiSCO-F. As it is evident from the table, DiSCO-F requires only one *MPI_Allreduce* of a vector of length n, whereas the DiSCO-S needs one *MPI_Allreduce* of a vector of length d and one broadcast (*MPI_Bcast*) of a vector of size d. So roughly speaking, when $n < d$ then DiSCO-F needs less communication. However, very interestingly, the advantage of DiSCO-F is the fact that it utilizes CPUs more efficiently. It also requires a less total amount of work to be performed on each node, leading to more balanced and efficient utilization of nodes (see Figure 1 for illustration. DiSCO-F utilizes resources more efficiently — Table 3 for shows the size of communication required in each PCG step).

Woodbury Formula for solving $Ps = r$

In each iteration of Algorithms 2 and 3, we need to solve a linear system in the form of $Ps = r$, where $P \in \mathbb{R}^{d\times d}$ in Algorithm 2 and $P \in \mathbb{R}^{d_i \times d_i}$ for $i = 1, 2, ..., m$ in Algorithm 3, which is usually very expensive. To solve it more efficiently, we can apply Woodbury Formula (Press et al. 2007).

Table 3: Comparison of communication between different algorithms.

DiSCO-S	DiSCO-F	DANE	CoCoA+
$2 \times \mathbb{R}^d$	$1 \times \mathbb{R}^n, 2 \times \mathbb{R}$	$2 \times \mathbb{R}^d$	$1 \times \mathbb{R}^d$

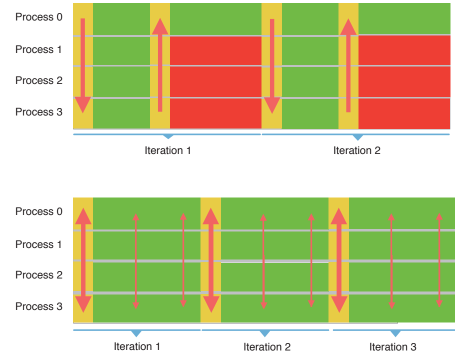

Figure 1: Flow diagrams of few iterations of DiSCO-S (top) and DiSCO-F (bottom). DiSCO-F uses less time for one iteration, due to the more efficient and balanced computation. Green boxes represent the processes are busy, while red boxes represent idle nodes. Yellow boxes show the status of communicating between all processes. Double arrows stand for *MPI_Allreduce* operations. The thin red arrows represent a communication of few scalars only.

Notice that if we use P defined in (3), P can be considered as sum of τ rank-1 updates on a diagonal matrix. For example, if $\phi(\cdot)$ is Quadratic Loss function, then

$$P = D + \tfrac{1}{\tau} \textstyle\sum_{i=1}^{\tau} x_i x_i^T. \tag{5}$$

If $\phi(\cdot)$ is Logistic Loss, then

$$P = D + \tfrac{1}{\tau} \textstyle\sum_{i=1}^{\tau} \frac{\exp(-w_k^T x_i)}{(\exp(-w_k^T x_i)+1)^2} x_i x_i^T. \tag{6}$$

In both cases, D is the diagonal matrix with $D_{ii} = \lambda + \mu$ for $i = 1, ..., m$. Then we can follow the procedure (Algorithm 4 to get the solution s. Note that $v \in \mathbb{R}^\tau$ and $\tau \ll d$ (in our experiments, $\tau = 100$ usually works very well). Therefore step 4 in Algorithm 4 can be done efficiently by any solver. In the numerical experiment section, we study the impact of τ on the performance of the final algorithm.

Algorithm 4 Woodbury Formula to solve $Ps = r$

1: Compute $z_i = \frac{1}{\lambda+\mu} x_i$ for $i = 1, ..., \tau$
2: Let $Z = [z_1, ..., z_\tau]$, $X = [x_1, ..., x_\tau]$
3: Compute $y = \frac{1}{\lambda+\mu} r$
4: Solve the linear system $(I + X^T Z)v = X^T y$
5: **Return**: $s = y - Xv$

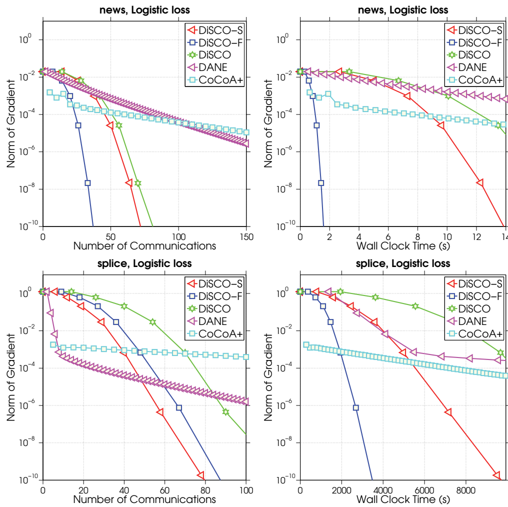

Figure 2: Norm of gradient vs. the round of communication, as well as norm of gradient vs. running time [s] for two datasets: news20 (top, $\lambda = 1e-3$), splice-site.test (bottom, $\lambda = 1e-6$).

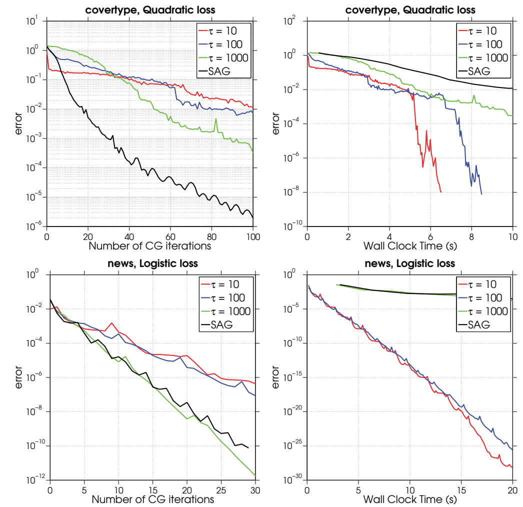

Figure 3: The effect of the choice of sample size when construction preconditioner matrix P. The y-axis shows the error$= \|Hv_k - \nabla f(w_k)\|_2$ for the linear system solving by Algorithm 3. The black lines represent the preconditioner proposed by (Zhang and Xiao 2015) and applying SAG. The other three lines stand for P in (5) and using Algorithm 4.

Numerical Experiments

We present experiments on four real-world datasets (two of them are large) distributed across multiple machines, running on the Amazon EC2 cluster. We show that DiSCO-F with a small τ converges to the optimal solution faster concerning the total rounds of communications compared to original DiSCO, DANE, and CoCoA+ in most cases. Also, for the dataset with $d > n$, DiSCO-F will also dominate others in wall-clock time.

We implement DiSCO and all other algorithms for comparison in C++, and run them on Amazon cluster using four m3.large EC2 instances. We apply all methods on Quadratic Loss and Logistic Loss for solving (P). A summary of the datasets used is shown in Table 4.

Comparison of different algorithms.

We compare the DiSCO-S, DiSCO-F, DiSCO, DANE and CoCOA+ directly using two datasets (new20 and splice-site.test) for Logistic Loss. In DiSCO-S and DiSCO-F, we set $\tau = 100$. In DiSCO and DANE, we apply Stochastic Average Gradient(SAG) (Schmidt, Roux, and Bach 2013) to solve the linear system $Ps = r$ or subproblems, respectively. Also, μ was set to $1e{-}2$ for both of them. In CoCoA+, SDCA was used as the solver for local subproblem.

Figure 2 shows the evolution of the norm of the gradient of the objective function as a function of number of communications and wall clock time respectively. In all the cases, DiSCO-F uses only half of the rounds of communications compared to DiSCO-S. Also, DiSCO-S often uses similar rounds of communications with the original DiSCO, which demonstrates the advantage of using a preconditioning matrix based on only a small subset of the samples. Finally, DANE and CoCoA+ will decrease the norm of gradient very fast at the first few iterations, but the decreasing become much weaker afterward.

Table 4: Datasets used for numerical experiments.

Dataset	n	d	size
a1a	1,605	119	112KB
covtype	581,012	54	70MB
news20	19,996	1,355,191	130MB
splice-site.test	4,627,840	11,725,480	273.4GB

For the news20 ($d \gg n$) and splice-site.test ($d \sim n$) dataset, the DiSCO-F converges to the optimal solution with fewer iterations than all the other methods. The elapsed time for DiSCO-F is only 10% of DiSCO-S for the news20 case. The explanation is that the size of vectors which has to be communicated is much smaller.

Impact of the Parameter τ.

In this section, we compare the performance of DiSCO-F algorithm under different preconditioners P. If we apply the method described in Algorithm 4, the parameter τ would determine how well the preconditioning matrix P can approximate the true Hessian H. In an extreme case, if we only use one machine and $\tau = n$, then $\|P - H\|_2 = 0$ and each iteration of Algorithm 1 will only use one iteration of PCG algorithm. However, too large τ will cause computation in Algorithm 4 to be expensive, thus resulting in long running time. Here we set $\tau \in \{10, 100, 1000\}$ and compare them to the preconditioner (very large τ) used in (Zhang and Xiao 2015) with SAG as the solver to solve the system $Ps = r$.

As shown in Figure 3, the larger τ we use, the less total number of communications the algorithm takes to reach an approximate solution. However, $\tau = 10$ or 100 always leads to much shorter time to convergence on these two datasets.

We also explore how different values of τ will affect the distribution of eigenvalues of the preconditioned Hessian matrix $P^{-1}H$. Figure 4 indicates that as τ becomes

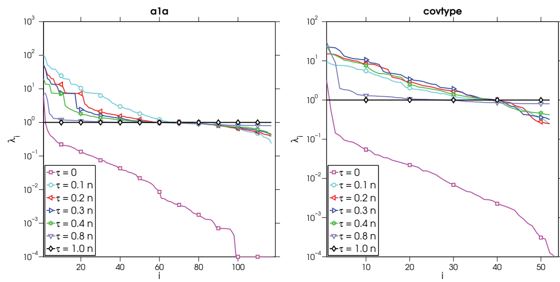

Figure 4: Sorted eigenvalues (from large ones to small ones) of preconditioned Hessian for different values of τ. The pink plot (when $\tau = 0$) shows the eigenvalues of the original Hessian. For all $\tau \neq 0$, the condition number of preconditioned Hessian has been reduced significantly.

larger, the condition number of $P^{-1}H$ will decrease to 1. Even if we use only 10% of data samples to construct P, the eigenvalues of $P^{-1}H$ are better concentrated, which leads to faster convergence when applying PCG.

Conclusion

We study inexact damped Newton method implemented in a distributed way based on DiSCO algorithm (Zhang and Xiao 2015). We found that partitioning the dataset by features leads to a decrease in the number of communications. Our algorithmic modifications lead to more balanced and better-performing algorithm. Also, the size of samples to generate the preconditioning matrix has been shrink from $\frac{n}{m}$ to $\tau \approx 100$, which greatly improves the efficiency of solving the linear system using PCG. Our experimental results show significant speedups over previous methods, including the original DiSCO algorithm as well as other state-of-the-art methods.

Acknowledgments

This research was supported by National Science Foundation grant CCF-1618717.

References

Agarwal, A., and Duchi, J. C. 2011. Distributed delayed stochastic optimization. In *NIPS*, 873–881.

Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning* 3(1):1–122.

Bradley, J. K.; Kyrola, A.; Bickson, D.; and Guestrin, C. 2011. Parallel coordinate descent for l1-regularized loss minimization. *arXiv:1105.5379*.

Defazio, A.; Bach, F.; and Lacoste-Julien, S. 2014. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, 1646–1654.

Dekel, O.; Gilad-Bachrach, R.; Shamir, O.; and Xiao, L. 2012. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research* 13(1):165–202.

Deng, W., and Yin, W. 2012. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing* 1–28.

Hsieh, C.-J.; Chang, K.-W.; Lin, C.-J.; Keerthi, S. S.; and Sundararajan, S. 2008. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, 408–415. ACM.

Jaggi, M.; Smith, V.; Takác, M.; Terhorst, J.; Krishnan, S.; Hofmann, T.; and Jordan, M. I. 2014. Communication-efficient distributed dual coordinate ascent. In *NIPS*, 3068–3076.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. *NIPS* 315–323.

Konečný, J.; Liu, J.; Richtárik, P.; and Takáč, M. 2014. mS2GD: Mini-batch semi-stochastic gradient descent in the proximal setting. *arXiv:1410.4744*.

Ma, C.; Smith, V.; Jaggi, M.; Jordan, M. I.; Richtárik, P.; and Takáč, M. 2015. Adding vs. averaging in distributed primal-dual optimization. In *ICML 2015 - Proceedings of the 32th International Conference on Machine Learning*, volume 37, 1973–1982. JMLR.

Marecek, J.; Richtárik, P.; and Takáč, M. 2014. Distributed block coordinate descent for minimizing partially separable functions. *Numerical Analysis and Optimization 2014, Springer Proceedings in Mathematics and Statistics*.

Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; and Flannery, B. P. 2007. Numerical recipes: the art of scientific computing.

Qu, Z.; Richtárik, P.; and Zhang, T. 2015. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *NIPS*, 865–873.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 693–701.

Richtárik, P., and Takáč, M. 2013. Distributed coordinate descent method for learning with big data. *arXiv:1310.2059*.

Rodgers, D. P. 1985. Improvements in multiprocessor system design. In *ACM SIGARCH Computer Architecture News*, volume 13, 225–231. IEEE Computer Society Press.

Schmidt, M.; Roux, N. L.; and Bach, F. 2013. Minimizing finite sums with the stochastic average gradient. *arXiv:1309.2388*.

Shalev-Shwartz, S.; Singer, Y.; Srebro, N.; and Cotter, A. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming* 127(1):3–30.

Shamir, O., and Srebro, N. 2014. Distributed stochastic optimization and learning. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, 850–857. IEEE.

Shamir, O.; Srebro, N.; and Zhang, T. 2013. Communication efficient distributed optimization using an approximate newton-type method. *arXiv:1312.7853*.

Takáč, M.; Bijral, A.; Richtárik, P.; and Srebro, N. 2013. Mini-batch primal and dual methods for SVMs. *ICML*.

Takáč, M.; Richtárik, P.; and Srebro, N. 2015. Distributed mini-batch SDCA. *arXiv:1507.08322*.

Yang, T.; Zhu, S.; Jin, R.; and Lin, Y. 2013. Analysis of distributed stochastic dual coordinate ascent. *arXiv:1312.1031*.

Yang, T. 2013. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, 629–637.

Zhang, Y., and Lin, X. 2015. DiSCO: Distributed Optimization for Self-Concordant Empirical Loss. In *ICML*, 362–370.

Zhang, Y., and Xiao, L. 2015. Communication-efficient distributed optimization of self-concordant empirical loss. *arXiv:1501.00263*.