Proceedings of Machine Learning Research vol 144:1–12, 2021

Accelerating Distributed SGD for Linear Regression using Iterative Pre-Conditioning

Kushal Chakrabarti KCHAK@TERPMAIL.UMD.EDU
University of Maryland, College Park, Maryland 20742, U.S.A.

Nirupam Gupta NIRUPAM.GUPTA@EPFL.CH
École polytechnique fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

Nikhil Chopra NCHOPRA@UMD.EDU
University of Maryland, College Park, Maryland 20742, U.S.A.

Abstract

This paper considers the multi-agent distributed linear least-squares problem. The system comprises multiple agents, each agent with a locally observed set of data points, and a common server with whom the agents can interact. The agents' goal is to compute a linear model that best fits the collective data points observed by all the agents. In the server-based distributed settings, the server cannot access the data points held by the agents. The recently proposed Iteratively Pre-conditioned Gradient-descent (IPG) method Chakrabarti et al. (2020a) has been shown to converge faster than other existing distributed algorithms that solve this problem. In the IPG algorithm, the server and the agents perform numerous iterative computations. Each of these iterations relies on the entire batch of data points observed by the agents for updating the current estimate of the solution. Here, we extend the idea of iterative pre-conditioning to the *stochastic* settings, where the server updates the estimate and the *iterative pre-conditioning matrix* based on a single randomly selected data point at every iteration. We show that our proposed Iteratively Pre-conditioned Stochastic Gradient-descent (IPSG) method converges linearly in expectation to a proximity of the solution. Importantly, we empirically show that the proposed IPSG method's convergence rate compares favorably to prominent stochastic algorithms for solving the linear least-squares problem in server-based networks.

Keywords: stochastic gradient descent; distributed systems; accelerated methods

1. Introduction

This paper considers solving the distributed linear least-squares problem using stochastic algorithms. In particular, as shown in in Fig. 1, we consider a server-based distributed system comprising m agents and a central server. The agents can only interact with the server, and the overall system is assumed synchronous. Each agent i has n *local* data points, represented by an *input matrix* A^i and an *output vector* B^i of dimensions $n \times d$ and $n \times 1$, respectively. Thus, for all $i \in \{1, \ldots, m\}$, $A^i \in \mathbb{R}^{n \times d}$ and $B^i \in \mathbb{R}^n$. For each agent i, we define a *local cost function* $F_i : \mathbb{R}^d \to \mathbb{R}$ such that for a given *regression parameter*

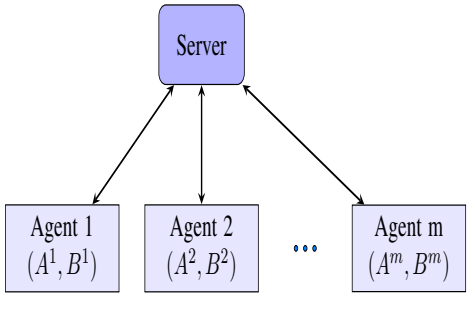

Figure 1: System architecture.

vector $x \in \mathbb{R}^d$,

$$F^i(x) = \frac{1}{2n} \left\| A^i x - B^i \right\|^2, \tag{1}$$

where $\|\cdot\|$ denotes the Euclidean norm. The agents' objective is to compute an optimal parameter vector $x^* \in \mathbb{R}^d$ such that

$$x^* \in \arg\min_{x \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} F^i(x). \tag{2}$$

Since each agent knows only a segment of the *collective data points*, they collaborate with the server for solving the distributed problem (2). However, the agents do not share their local data points with the server. An algorithm that enables the agents to jointly solve the above problem in the architecture of Fig. 1 without sharing their data points is defined as a *distributed algorithm*.

There are several theoretical and practical reasons for solving the distributed problem (2) using stochastic methods rather than batched optimization methods, particularly when the number of data-points is abundant Bottou et al. (2018). The basic prototype of the stochastic optimization methods that solve (2) is the traditional stochastic gradient (SGD) Bottou et al. (2018). Several accelerated variants of the stochastic gradient descent algorithm have been proposed in the past decade Duchi et al. (2011); Kingma and Ba (2014); Zeiler (2012); Tieleman and Hinton (2012); Reddi et al. (2019); Dozat (2016). A few of such well-known methods are the adaptive gradient descent (AdaGrad) Duchi et al. (2011), adaptive momentum estimation (Adam) Kingma and Ba (2014), AMSGrad Reddi et al. (2019). These algorithms are stochastic, wherein the server maintains an estimate of a solution defined by (2), which is refined iteratively by the server using the *stochastic gradients* computed by a randomly chosen agent.

In particular, Adam has been demonstrated to compare favorably with other stochastic optimization algorithms for a wide range of optimization problems Radford et al. (2015); Peters et al. (2018); Wu et al. (2016). However, Adam updates the current estimate effectively based on only a window of the past gradients due to the exponentially decaying term present in its estimate updating equation, which leads to poor convergence in many problems Reddi et al. (2019). A recently proposed variant of Adam is the AMSGrad algorithm, which proposes to fix Adam's convergence issue by incorporating "long-term memory" of the past gradients.

In this paper, we propose a *stochastic iterative pre-conditioning* technique for improving the rate of convergence of the distributed stochastic gradient descent method when solving the linear least-squares problem (2) in distributed networks. The idea of *iterative pre-conditioning* in the deterministic (batched data) case has been first proposed in Chakrabarti et al. (2020a), wherein the server updates the estimate using the sum of the agents' gradient multiplied with a suitable iterative pre-conditioning matrix. Updating the pre-conditioning matrix depends on the entire dataset at each iteration. The proposed algorithm extends that idea to the *stochastic* settings, where the server updates both the estimate and the iterative pre-conditioning matrix based on a randomly chosen agents' *stochastic gradient* at every iteration. Each agent computes its stochastic gradient based on a single randomly chosen data point from its local set of data points. Using real-world datasets, we empirically show that the proposed algorithm converges in fewer iterations compared to the aforementioned state-of-the-art distributed methods.

We note that the prior work on the formal convergence of the iteratively pre-conditioned gradient-descent method only considers the batched data at every iteration Chakrabarti et al. (2020a). There-

fore, besides empirical results, we also present a formal analysis of the proposed algorithm's convergence in stochastic settings.

1.1. Summary of our contributions

1. We present a formal convergence analysis of our proposed algorithm. Our convergence result can be informally summarized as follows. *Suppose the solution of problem* (2) *is unique, and the variances of the stochastic gradients computed by the agents are bounded. In that case, our proposed algorithm, i.e., Algorithm 1, converges linearly in expectation to a proximity of the solution of the problem* (2). The approximation error is proportional to the algorithm's stepsize and the variances of the stochastic gradients. Note that, as shown in prior work Chakrabarti et al. (2020c), our algorithm converges *superlinearly* to the exact solution when the gradient noise is zero.

   Formal details are presented in Theorem 1 in Section 3.2.

2. Using real-world datasets, we empirically show that our proposed algorithm's convergence rate is superior to that of the state-of-the-art stochastic methods when distributively solving linear least-squares problems. These datasets comprise

   - four benchmark datasets from the SuiteSparse Matrix Collection;
   - a subset of the *"cleveland"* dataset from the UCI Machine Learning Repository, which contains binary classification data of whether the patient has heart failure or not based on 13 features;
   - a subset of the *"MNIST"* dataset for classification of handwritten digits one and five.

   Please refer to Section 4 for further details.

2. SGD with Iterative Pre-Conditioning

In this section, we present our algorithm. Our algorithm follows the basic prototype of the stochastic gradient descent method in distributed settings. However, unlike the traditional distributed stochastic gradient descent, the server in our algorithm multiplies the stochastic gradients received from the agents by a *stochastic pre-conditioner* matrix. These pre-conditioned stochastic gradients are then used by the server to update the current estimate. In literature, this technique is commonly referred as *pre-conditioning* Nocedal and Wright (2006). It should be noted that unlike the conventional pre-conditioning techniques Fessler, in our case, the stochastic pre-conditioning matrix itself is updated over the iterations with help from the agents. Hence, the name *iterative pre-conditioning*.

In order to present the algorithm, we introduce some notation. The *individual data points* of the agents are represented by an *input row vector* a of dimensions $1 \times d$ and a *scalar output* b. Thus, $a \in \mathbb{R}^{1\times d}$ and $b \in \mathbb{R}$. For each data point (a, b), we define *individual cost function* $f : \mathbb{R}^d \to \mathbb{R}$ such that for a given $x \in \mathbb{R}^d$,

$$f(x; a, b) = \frac{1}{2} (a\, x - b)^2 \,, \tag{3}$$

and the gradient of the individual cost function f as

$$g(x; a, b) = \nabla_x f(x; a, b) = a^T \ (a\, x - b) \,. \tag{4}$$

Here, $(\cdot)^T$ denotes the transpose.

In each iteration $t \in \{0, 1, \ldots\}$, the server maintains an estimate $x(t)$ of a minimum point (2), and a stochastic pre-conditioner matrix $K(t) \in \mathbb{R}^{d \times d}$. The initial estimate $x(0)$ and the pre-conditioner matrix $K(0)$ are chosen arbitrarily from $\mathbb{R}^d$ and $\mathbb{R}^{d \times d}$, respectively. For each iteration $t = 0, 1, \ldots$, the algorithm steps are presented below.

2.1. Steps in each iteration t

Before initiating the iterations, the server chooses a positive scalar real-valued parameter β and broadcast it to all the agents. We number the agents in order from 1 to m. In each iteration t, the proposed algorithm comprises of four steps described below. These steps are executed collaboratively by the server and the agents, without requiring any agent to share its local data points. For each iteration t, the server also chooses two positive scalar real-valued parameters α and δ.

- *Step 1:* The server sends the estimate $x(t)$ and the pre-conditioner matrix $K(t)$ to each agent $i \in \{1, \ldots, m\}$.

- *Step 2:* Each agent $i \in \{1, \ldots, m\}$ chooses a data point (a^{i_t}, b^{i_t}) *uniformly* at random from its local data points (A^i, B^i). Note that, a^{i_t} and b^{i_t} are respectively a row in the input matrix A^i and the output vector B^i of agent i. Each data point is independently and identically distributed (i.i.d.). Based on the selected data point (a^{i_t}, b^{i_t}), each agent i then computes a stochastic gradient, denoted by $g^{i_t}(t)$, which is defined as

$$g^{i_t}(t) = g(x(t); a^{i_t}, b^{i_t}). \tag{5}$$

  In the same step, each agent $i \in \{1, \ldots, m\}$ computes a set of vectors $\left\{h_j^{i_t}(t) : \ j = 1, \ldots, \ d\right\}$:

$$h_j^{i_t}(t) = h_j(k_j(t); a^{i_t}, b^{i_t}), \tag{6}$$

  where the function $h_j : \mathbb{R}^d \to \mathbb{R}^d$ is defined below. Let I denote the $(d \times d)$-dimensional identity matrix. Let e_j and $k_j(t)$ denote the j-th columns of matrices I and $K(t)$, respectively. For each column $j \in \{1, \ldots, d\}$ of $K(t)$ and each individual data point (a, b), we define

$$h_j(k_j; a, b) = \left(a^T a + \beta I\right) k_j - e_j. \tag{7}$$

- *Step 3:* Each agent $i \in \{1, \ldots, m\}$ sends the stochastic gradient $g^{i_t}(t)$ and the set of stochastic vectors $\left\{h_j^{i_t}(t), \ j = 1, \ldots, \ d\right\}$ to the server.

- *Step 4:* The server draws an i.i.d. sample ζ_t *uniformly* at random from the set of agents $\{1, \ldots, m\}$ and updates the matrix $K(t)$ to $K(t+1)$ such that, for each $j \in \{1, \ldots, d\}$,

$$k_j(t+1) = k_j(t) - \alpha h_j^{\zeta_t}(t). \tag{8}$$

  Finally, the server updates the estimate $x(t)$ to $x(t+1)$ such that

$$x(t+1) = x(t) - \delta K(t+1) g^{\zeta_t}(t). \tag{9}$$

  Parameter δ is a non-negative real value, commonly referred as the *stepsize*.

These steps of our algorithm are summarized in Algorithm 1.

Algorithm 1 *Iteratively Pre-Conditioned Stochastic Gradient-descent (IPSG).*

1: The server initializes $x(0) \in \mathbb{R}^d$, $K(0) \in \mathbb{R}^{d\times d}$, $\beta > 0$ and chooses $\{\alpha > 0, \delta > 0 : \ t = 0, 1, \ldots\}$.
2: **Steps in each iteration** $t \in \{0,\ 1,\ 2, \ldots\}$:
3: The server sends $x(t)$ and $K(t)$ to all the agents.
4: Each agent $i \in \{1, \ldots,\ m\}$ *uniformly* selects an i.i.d. data point (a^{i_t}, b^{i_t}) from its local data points (A^i, B^i).
5: Each agent $i \in \{1, \ldots,\ m\}$ sends to the server a stochastic gradient $g^{i_t}(t)$, defined in (5), and d stochastic vectors $h_1^{i_t}(t), \ldots,\ h_d^{i_t}(t)$, defined in (6).
6: The server *uniformly* draws an i.i.d. sample ζ_t from the set of agents $\{1, \ldots, m\}$.
7: The server updates $K(t)$ to $K(t+1)$ as defined by (8).
8: The server updates the estimate $x(t)$ to $x(t+1)$ as defined by (9).

3. Convergence Analysis

In this section, we present the formal convergence guarantees of Algorithm 1. We begin by introducing some notation and our main assumptions.

3.1. Notation and assumptions

Table 1: *Additional Notation.*

$A = \left[(a^1)^T, \ldots,\ (a^N)^T\right]^T$	$K_\beta = \left(\frac{1}{N}A^TA + \beta I\right)^{-1}$
$C_1 = \max_i \left\|\left(a^i\right)^T a^i - \frac{1}{N}A^TA\right\|$	$\rho = \frac{1}{N}\sum_{i=1}^N \left\|I - \alpha\left(\left(a^i\right)^T a^i + \beta I\right)\right\|$
$s_1 \geq \ldots \geq s_d \geq 0$: eigenvalues of the positive semi-definite matrix A^TA	
Λ_i and λ_i: the largest and the smallest eigenvalue of $\left(a^i\right)^T a^i$	$L = \beta + \max_{i=1,\ldots,N}\Lambda_i$
$\mu = \left(1 - \frac{2\alpha s_d}{N}(1-\alpha L)\right)$	$\sigma^2 = \max\limits_{j=1,\ldots,d}\frac{1}{N}\sum\limits_{i=1}^N \left\|\left(\left(a^i\right)^T a^i + \beta I\right)K_\beta\, e_j - e_j\right\|^2$
$C_3 = \frac{\alpha N\sigma^2}{s_d(1-\alpha L)}$	$C_2 = \frac{\alpha}{N}\sum_{i=1}^N \left\|\left(a^i\right)^T a^i - \frac{1}{N}A^TA\right\| \|K_\beta\|$
$\varrho = \left\|I - \alpha\left(\frac{1}{N}A^TA + \beta I\right)\right\|$	$C_5(t) = 2C_1E_2\frac{s_1}{N}\left(\|K_\beta\| + \left\|\widetilde{K}(0)\right\|\varrho^t\right)$
$C_6(t) = \frac{2s_d}{s_d+N\beta} - 2\frac{s_1}{N}\left\|\widetilde{K}(0)\right\|\varrho^{t+1}$	$C_7(t) = 2C_1E_1\left(\|K_\beta\| + \left\|\widetilde{K}(0)\right\|\varrho^t\right)$
$C_8(t) = (V_2+1)\frac{s_1^2}{N}(dC_3 + \|K_\beta\|^2 + 2C_2\|K_\beta\|\sum_{j=0}^t\rho^j + \left\|\widetilde{K}(0)\right\|_F^2\mu^{t+1}$ $+2\|K_\beta\|\left\|\widetilde{K}(0)\right\|\rho^{t+1}) + 0.5$	
$z(t) = x(t) - x^*$	$R_1(t) = 1 + \delta^2C_8(t) + \alpha\delta C_5(t) - \delta C_6(t)$
$R_2(t) = \delta^2V_1N(dC_3 + \|K_\beta\|^2 + 2C_2\|K_\beta\|\sum_{j=0}^t\rho^j + \left\|\widetilde{K}(0)\right\|_F^2\mu^{t+1}$ $+2\|K_\beta\|\left\|\widetilde{K}(0)\right\|\rho^{t+1}) + \frac{1}{2}\alpha^2C_7(t)^2$	

The *collective input matrix* A, and the *collective output vector* B are defined to be

$$A = \left[(A^1)^T, \ldots, (A^m)^T\right]^T, \; B = \left[(B^1)^T, \ldots, (B^m)^T\right]^T. \tag{10}$$

Define $N = mn$. Note that, $A \in \mathbb{R}^{N \times d}$ and $B \in \mathbb{R}^N$.

For each iteration $t \geq 0$ we define the following.

- Let $\mathbb{E}_{\zeta_t}[\cdot]$ and for each agent $i \in \{1, \ldots, m\}$ $\mathbb{E}_{i_t}[\cdot]$ denote the conditional expectation of a function the random variables ζ_t and i_t, respectively, given the current estimate $x(t)$ and the current pre-conditioner $K(t)$.
- Let $I_t = \{i_t, \; i = 1, \ldots, m\} \cup \{\zeta_t\}$ and $\mathbb{E}_{I_t}[\cdot] = \mathbb{E}_{1_t, \ldots m_t, \zeta_t}(\cdot)$.
- Let $\mathbb{E}_t[\cdot]$ denote the total expectation of a function of the random variables $\{I_0, \ldots, I_t\}$ given the initial estimate $x(0)$ and initial pre-conditioner matrix $K(0)$. Specifically,

$$\mathbb{E}_t[\cdot] = \mathbb{E}_{I_0, \ldots, I_t}(\cdot), \; t \geq 0. \tag{11}$$

- Define the conditional variance of the stochastic gradient $g^{i_t}(t)$, which is a function of the random variable i_t, given the current estimate $x(t)$ and the current pre-conditioner $K(t)$ as

$$\mathbb{V}_{i_t}\left[g^{i_t}(t)\right] = \mathbb{E}_{i_t}\left[\left\|g^{i_t}(t) - \mathbb{E}_{i_t}\left[g^{i_t}(t)\right]\right\|^2\right] = \mathbb{E}_{i_t}\left[\left\|g^{i_t}(t)\right\|^2\right] - \left\|\mathbb{E}_{i_t}\left[g^{i_t}(t)\right]\right\|^2. \tag{12}$$

Additional notation are listed in Table 1. Among these notation, K_β is an approximation of the inverse Hessian matrix, to which the sequence of the pre-conditioner matrices converges in expectation. $R_1(t)$ and $R_2(t)$ characterize the estimation error after t iterations. The other notation in Table 1 are required to define $R_1(t)$ and $R_2(t)$.

We make the following assumption on the rank of the matrix $A^T A$.

Assumption 1 *Assume that the matrix $A^T A$ is full rank.*

Note that Assumption 1 holds true if and only if the matrix $A^T A$ is positive definite with $s_d > 0$. As the Hessian of the aggregate cost function $\sum_{i=1}^m F^i(x)$ is equal to $A^T A$ for all x (see (1), under Assumption 1, the aggregate cost function has a unique minimum point. Equivalently, the solution of the distributed least squares problem defined by (2) is unique.

We also assume, as formally stated in Assumption 2 below, that the variance of the stochastic gradient for each agent is bounded. This is a standard assumption for the analysis of stochastic algorithms Bottou et al. (2018).

Assumption 2 *Assume that there exist two non-negative real scalar values V_1 and V_2 such that, for each iteration $t = 0, 1, \ldots$ and each agent $i \in \{1, \ldots, m\}$,*

$$\mathbb{V}_{i_t}\left[g^{i_t}(t)\right] \leq V_1 + V_2 \left\|\textstyle\sum_{i=1}^m \nabla F^i(x(t))/m\right\|^2.$$

Next, we present our key result on the convergence of Algorithm 1. The proof of Theorem 1 can be found in Chakrabarti et al. (2020b).

3.2. Main Theorem

Theorem 1 *Consider Algorithm 1 with parameters* $\beta > 0$, $\alpha < \min\left\{\frac{N}{s_d}, \frac{1}{L}, \frac{2}{(s_1/N)+\beta}\right\}$ *and* $\delta > 0$. *If Assumptions 1 and 2 are satisfied, then there exist two non-negative real scalar values* $E_1 \geq \sqrt{V_1 N}$ *and* $E_2 \geq \sqrt{V_2 N}$ *such that the following statements hold true.*

1. *If the stepsize* δ *is sufficiently small, then there exists a non-negative integer* $T < \infty$ *such that for any iteration* $t \geq T$, $R_1(t)$ *is positive and less than* 1.

2. *For an arbitrary time step* $t \geq 0$, *given the estimate* $x(t)$ *and the matrix* $K(t)$,

$$\mathbb{E}_t\left[\|z(t+1)\|^2\right] \leq R_1(t)\,\|z(t)\|^2 + R_2(t).$$

3. *Given arbitrary choices of the initial estimate* $x(0) \in \mathbb{R}^d$ *and the pre-conditioner matrix* $K(0) \in \mathbb{R}^{d\times d}$,

$$\lim_{t\to\infty} \mathbb{E}_t\left[\|z(t+1\|^2\right] \leq \delta^2 V_1 N\left(dC_3 + \|K_\beta\|^2 + \frac{2C_2\,\|K_\beta\|}{1-\rho}\right) + 2\alpha^2\left(C_1 E_1\,\|K_\beta\|\right)^2.$$

The implications of Theorem 1 are as follows.

- According to Part (1) and (2) of Theorem 1, for small enough values of the parameters α and stepsize δ, as $R_1(t) \in (0,1)$ after some finite iterations, Algorithm 1 converges *linearly* in expectation to a neighborhood of the minimum point x^* of the least-squares problem (2).

- According to Part (3) of Theorem 1, the neighborhood of x^*, to which the estimates of Algorithm 1 converges in expectation, is $\mathcal{O}(V_1)$. In other words, the sequence of expected "distance" between the minima x^* of (2) and the final estimated value of Algorithm 1 is proportional to the variance of the stochastic gradients at the minimum point.

4. Experimental Results

In this section, we present our experimental results validating the convergence of our proposed algorithm on real-world problems and its comparison with related methods.

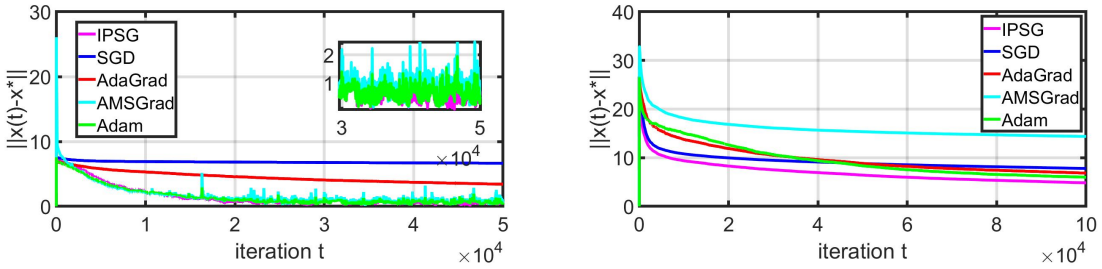

Figure 2: Estimation error $\|x(t) - x^*\|$ of different algorithms for (a)*"MNIST"* and (b)*"illc1850"*. For all the algorithms, (a): $x(0) = [10, \ldots, 10]^T$; (b)-(f): $x(0) = [0, \ldots, 0]^T$. Additionally, for IPSG, $K(0) = O_{d\times d}$. The other parameters are enlisted in Table 2.

Table 2: *The parameters used in different algorithms.*

Dataset	IPSG	SGD Bottou et al. (2018)	AdaGrad Duchi et al. (2011)	AMSGrad Reddi et al. (2019)	Adam Kingma and Ba (2014)
cleveland	$\alpha = 0.0031$, $\delta = 0.5$, $\beta = 30$	$\alpha = 0.0031$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha = 0.05$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$	$\alpha = 0.05$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
abtaha1	$\alpha = 0.0052$, $\delta = 2$, $\beta = 5$	$\alpha = 0.0052$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{1}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.5}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
abtaha2	$\alpha = 0.0033$, $\delta = 2$, $\beta = 5$	$\alpha = 0.0033$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha = 1$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.5}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
MNIST	$\alpha = 0.0003$, $\delta = 0.1$, $\beta = 1$	$\alpha = 0.0003$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha = 1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$	$\alpha = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
gre_343	$\alpha = 1.2$, $\delta = 2.5$, $\beta = 0.5$	$\alpha = 1.96$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.1}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.2}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
illc1850	$\alpha = 0.4436$, $\delta = 2$, $\beta = 1$	$\alpha = 0.4436$	$\alpha = 1$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.5}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-7}$	$\alpha_t = \frac{0.5}{\sqrt{t}}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$

We conduct experiments for different collective data points (A, B). Four of these datasets are from the the benchmark datasets available in SuiteSparse Matrix Collection Davis and Hu (2011). Particularly these four datasets are *"abtaha1"*, *"abtaha2"*, *"gre_343"*, and *"illc1850"*. The fifth dataset, *"cleveland"*, is from the UCI Machine Learning Repository Dua and Graff (2017). The sixth and final dataset is the *"MNIST"* MNI dataset.

In the case of the first four aforementioned datasets, the problem is set up as follows. Consider a particular dataset *"abtaha1"*. Here, the matrix A has 14596 rows and $d = 209$ columns. The collective output vector B is such that $B = Ax^*$ where x^* is a 209 dimensional vector, all of whose entries are unity. The data points represented by the rows of the matrix A and the corresponding observations represented by the elements of the vector B are divided amongst $m = 4$ agents numbered from 1 to 4. Since the matrix A for this particular dataset has 14596 rows and 209 columns, each of the four agents $1, \ldots, 4$ has a data matrix A^i of dimension 3649×209 and a observation vector B^i of dimension 3649. The data points for the other three datasets, *"abtaha2"*, *"gre_343"*, and *"illc1850"*, are similarly distributed among $m = 4$, $m = 7$ and $m = 10$ agents, respectively.

For the fifth dataset, 212 arbitrary instances from the *"cleveland"* dataset have been selected. This dataset has 13 numeric attributes, each corresponding to a column in the matrix A, and a target class (whether the patient has heart disease or not), which corresponds to the output vector B. Since the attributes in the matrix A has different units, its each column is shifted by the mean value of the corresponding column and then divided by the standard deviation of that column. Finally, a 212-dimensional column vector of unity is appended to this pre-processed matrix. This is our final input matrix A of dimension (212×14). The collective data points (A, B) are then distributed among $m = 4$ agents, in the manner described earlier.

From the training examples of the *"MNIST"* dataset, we select 1500 arbitrary instances labeled as either the digit one or the digit five. For each instance, we calculate two quantities, namely the average intensity of an image and the average symmetry of an image Abu-Mostafa et al. (2012). Let the column vectors a_1 and a_2 respectively denote the average intensity and the average symmetry of those 1500. Then, our input matrix before pre-processing is $\begin{bmatrix} a_1 & a_2 & a_1.^2 & a_1 .* a_2 & a_2.^2 \end{bmatrix}$. Here, $(.*)$ represents element-wise multiplication and $(.^2)$ represents element-wise squares. This raw input matrix is then pre-processed as described earlier for the *"cleveland"* dataset. Finally, a 1500-dimensional column vector of unity is appended to this pre-processed matrix. This is our final input matrix A of dimension (1500×6). The collective data points (A, B) are then distributed among $m = 10$ agents, in the manner already described for the other datasets.

As the matrix $A^T A$ is positive definite in each of these cases, the optimization problem (2) has a unique solution x^* for all of these datasets.

We compare the performance of our proposed IPSG method (Algorithm 1) on the aforementioned datasets, with the other stochastic algorithms mentioned in Section 1. Specifically, these algorithms are stochastic gradient descent (SGD) Bottou et al. (2018), adaptive gradient descent (AdaGrad) Duchi et al. (2011), adaptive momentum estimation (Adam) Kingma and Ba (2014), and AMSGrad Reddi et al. (2019) in the distributed network architecture of Fig. 1. These algorithms are implemented with different combinations of the respective algorithm parameters on the individual datasets. The parameter combinations are described below.

IPSG: The optimal (smallest) convergence rate of the deterministic version of the proposed IPSG method is obtained when $\alpha = \frac{2}{s_1+s_d}$ Chakrabarti et al. (2020c). For each of the six datasets, we find that the IPSG method converges fastest when the parameter α is set similarly as $\alpha = \frac{2}{s_1+s_d}$. The stepsize parameter δ of the IPSG algorithm is chosen from the set $\{0.1, 0.5, 1, 2, 2.5\}$. The parameter β is chosen from the set $\{0.1, 0.5, 1, 5, 10, 30, 50\}$.

SGD: The SGD algorithm has only one parameter: the stepsize, denoted as α Bottou et al. (2018). The deterministic version of the SGD method is the gradient-descent method, which has the optimal rate of convergence when $\alpha = \frac{2}{s_1+s_d}$. We find that the SGD method converges fastest when the stepsize parameter is similarly set as $\alpha = \frac{2}{s_1+s_d}$. s_1 and s_d depends on the collective data matrix A, and their values may not be known to the server. When the actual values or estimates of s_1 and s_d are not known, the parameter α in both the IPSG and the SGD algorithm can be experimentally set by trying several values of different orders, as done for the parameters δ and β in the IPSG method.

AdaGrad: The stepsize parameter α of the AdaGrad algorithm Duchi et al. (2011) is selected from the set $\{1, 0.1, \frac{1}{t}\}$. The parameter ϵ is set at its usual value of 10^{-7}.

Adam and **AMSGrad**: The stepsize α Kingma and Ba (2014); Reddi et al. (2019) is selected from the set $\{c, \frac{c}{\sqrt{t}}\}$ where c is from the set $\{1, 0.5, 0.1, 0.2, 0.05, 0.01\}$. The parameter β_1 is set at its usual value of 0.9. The parameter β_2 is selected from their usual values of $\{0.99, 0.999\}$. The

parameter ϵ is set at 10^{-7}. The best parameter combinations from above, for which the respective algorithms converge in a fewer number of iterations, are reported for each dataset in Table 2.

The initial estimate $x(0)$ for all of these algorithms is chosen as the d-dimensional zero vector for each dataset except the *"cleveland"* dataset. For *"cleveland"* dataset, $x(0)$ is chosen as the d-dimensional vector whose each entry is 10. The initial pre-conditioner matrix $K(0)$ for the IPSG algorithm is the zero matrix of dimension $(d \times d)$.

Table 3: *Comparisons between the number of iterations required by different algorithms to attain the specified values for the relative estimation errors* $\epsilon_{tol} = \|x(t) - x^*\| \,/\, \|x(0) - x^*\|$.

Dataset	$\kappa(A^TA)$	ϵ_{tol}	IPSG	SGD	AdaGrad	AMSGrad	Adam
cleveland	7.34	1.5×10^{-3}	4.11×10^3	4.71×10^3	6.04×10^3	3.63×10^3	4.11×10^3
abtaha1	1.5×10^2	10^{-3}	7.35×10^4	$> 10^5$	9.75×10^4	$> 10^5$	$> 10^5$
abtaha2	1.5×10^2	2×10^{-3}	9.86×10^4	$> 10^5$	$> 10^5$	7.6×10^4	$> 10^5$
MNIST	2.59×10^3	2.6×10^{-3}	3.41×10^4	$> 5 \times 10^4$	$> 5 \times 10^4$	$> 5 \times 10^4$	4.41×10^4
gre_343	1.25×10^4	4×10^{-3}	3.88×10^4	4.43×10^5	$> 10^5$	$> 10^5$	$> 10^5$
illc1850	1.93×10^6	0.2	8.06×10^4	3.31×10^5	2.81×10^5	$> 5 \times 10^5$	1.63×10^5

We compare the number of iterations needed by these algorithms to reach a *relative estimation error* defined as $\epsilon_{tol} = \frac{\|x(t)-x^*\|}{\|x(0)-x^*\|}$. Each iterative algorithm is run (ref. Fig. 2) until its relative estimation error does not exceed ϵ_{tol} over a period of 10 consecutive iterations, and the smallest such iteration is reported in Table 3. The specific values of the algorithm parameters are tabulated in Table 2 for all six datasets. The second column of Table 3 indicates the condition number $\kappa(A^TA)$ for each dataset. From Table 3, we see that the IPSG algorithm converges fastest among the algorithms on four out of the six datasets, except for *"cleveland"* and *"abtaha2"*. Note that these two datasets have small condition number of order 10 and 10^2. Even for these two datasets, only the AMSGrad algorithm requires fewer iterations than IPSG. Moreover, from the datasets *"MNIST"*, *"gre_343"*, and *"illc1850"*, we observe that the differences between the proposed IPSG method and the other methods are significant when the condition number of the matrix A^TA is of order 10^3 or more. Thus, our claim on improvements over the prominent stochastic algorithms for solving the distributed least-squares problem (2) is corroborated by the above experimental results.

Acknowledgments

This work is being carried out as a part of the Pipeline System Integrity Management Project, which is supported by the Petroleum Institute, Khalifa University of Science and Technology, Abu Dhabi, UAE. Nirupam Gupta was sponsored by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.

References

MNIST in CSV. https://www.kaggle.com/oddrationale/mnist-in-csv. Accessed: 19-September-2020.

Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Kushal Chakrabarti, Nirupam Gupta, and Nikhil Chopra. Iterative pre-conditioning to expedite the gradient-descent method. In *2020 American Control Conference (ACC)*, pages 3977–3982, 2020a.

Kushal Chakrabarti, Nirupam Gupta, and Nikhil Chopra. Accelerating distributed SGD for linear least-squares problem. *arXiv preprint arXiv:2011.07595*, 2020b.

Kushal Chakrabarti, Nirupam Gupta, and Nikhil Chopra. Iterative pre-conditioning for expediting the gradient-descent method: The distributed linear least-squares problem. *arXiv preprint arXiv:2008.02856*, 2020c.

Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Dheeru Dua and Casey Graff. UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/Heart+Disease, 2017. University of California, Irvine, School of Information and Computer Sciences.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Jeffrey A Fessler. Image reconstruction: Algorithms and analysis. http://web.eecs.umich.edu/~fessler/book/c-opt.pdf. [Online book draft; accessed 11-November-2020].

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.