Machine Learning (2023) 112:3773–3804
https://doi.org/10.1007/s10994-021-06001-x

Byzantine-robust distributed sparse learning for *M*-estimation

Jiyuan Tu[1] · **Weidong Liu**[2] · **Xiaojun Mao**[3]

Received: 6 June 2020 / Revised: 14 March 2021 / Accepted: 14 May 2021 /
Published online: 26 July 2021

Abstract
In a distributed computing environment, there is usually a small fraction of machines that are corrupted and send arbitrary erroneous information to the master machine. This phenomenon is modeled as a Byzantine failure. Byzantine-robust distributed learning has recently become an important topic in machine learning research. In this paper, we develop a Byzantine-resilient method for the distributed sparse *M*-estimation problem. When the loss function is non-smooth, it is computationally costly to solve the penalized non-smooth optimization problem in a direct manner. To alleviate the computational burden, we construct a pseudo-response variable and transform the original problem into an ℓ_1-penalized least-squares problem, which is much more computationally feasible. Based on this idea, we develop a communication-efficient distributed algorithm. Theoretically, we show that the proposed estimator obtains a fast convergence rate with only a constant number of iterations. Furthermore, we establish a support recovery result, which, to the best of our knowledge, is the first such result in the literature of Byzantine-robust distributed learning. We demonstrate the effectiveness of our approach in simulation.

Keywords Byzantine robustness · *M*-estimation · Median-of-means · Support recovery

Weidong Liu and Xiaojun Mao are the co-corresponding authors.

Editors: Daniel Fremont, Mykel Kochenderfer, Alessio Lomuscio, Dragos Margineantu, Cheng Soon-Ong.

✉ Weidong Liu
weidongl@sjtu.edu.cn

✉ Xiaojun Mao
maoxj@fudan.edu.cn

Jiyuan Tu
tujy.19@gmail.com

1 School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai, China

2 School of Mathematical Sciences - School of Life Sciences and Biotechnology - MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China

3 School of Data Science, Fudan University, Shanghai, China

1 Introduction

With the development of modern technology, unprecedented data sizes are generated in many fields of scientific studies, and that creates the need for statistical analysis. The computational power of a single computer is no longer sufficient to store and process modern data sets. To address the problem of storage and computation, several distributed computing methods have been proposed. In a distributed system, data are partitioned across multiple machines. Among them, there is a single master machine in charge of updating and broadcasting the parameters. The rest of the machines, called worker machines, maintain a large part of the data and can only communicate with the master machine. Due to the vulnerability of modern machine learning systems, robust distributed learning has recently become an important topic in machine learning research. In particular, in a large distributed data processing system, there is usually a small fraction of worker machines that sends arbitrary erroneous information to the master machine due to a system breakdown or hacker attack. This phenomenon is typically modeled as Byzantine failure (Lamport et al. 1982). A Byzantine-tolerant distributed statistical method is referred to as conducting a good estimate for the parameters we care about, even with the presence of a moderate fraction of Byzantine machines in the distributed system.

Byzantine-robust distributed learning has an intimate connection with robust estimation in the statistics literature (Huber 2004). In fact, in the field of machine learning, a popular approach to hedge against Byzantine failure is to take the median (Xie et al. 2018; Yin et al. 2018, 2019), instead of the vanilla sample mean, among the data transmitted to the master machine. It has been well known that the median estimator and its variants are commonly used in robust statistics (see, *e.g.*, Minsker 2015; Lecué and Lerasle 2020; Lugosi and Mendelson 2019; Minsker 2019).

In this paper, we are interested in a special class of stochastic optimization problems: the sparse *M*-estimation. Let $\boldsymbol{X} = (X_1, X_2, \dots, X_p)^{\mathrm{T}} \in \mathbb{R}^p$ be a p-dimensional covariate, and Y be the response variable. Our target is to estimate the true parameter $\boldsymbol{\beta}^*$ defined in the following *M*-estimation problem

$$\boldsymbol{\beta}^* = \mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\mathbb{E}\{\mathcal{L}(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})\}, \tag{1}$$

where $\mathcal{L}(\cdot)$ is a pre-determined convex loss function. The vector $\boldsymbol{\beta}^*$ is assumed to be sparse, i.e. the number of nonzero entries in $\boldsymbol{\beta}^*$ is small compared with p. In practice, we cannot minimize the population loss function (1) directly. Instead, one may minimize the empirical loss function with an l_1-penalty to approximate the true parameter $\boldsymbol{\beta}^*$,

$$\mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|_1, \tag{2}$$

where $(Y_i, \boldsymbol{X}_i)$, $1 \le i \le N$ are i.i.d. observations and $|\boldsymbol{\beta}|_1 = \sum_{i=1}^{p}|\beta_i|$ with $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^{\mathrm{T}}$; see (Tibshirani 1996; Zhao and Yu 2006; Wainwright 2009; Bühlmann and Van De Geer 2011; Hastie et al. 2015). In this paper, we consider the distributed estimation of $\boldsymbol{\beta}^*$, in which the N samples are evenly stored in $m+1$ machines $\mathcal{H}_0, \dots, \mathcal{H}_m$ (*i.e.*, each local machine has n samples and $N = (m+1)n$). Here we denote $\mathcal{H}_0$ as the master machine, which is in charge of collecting information from local machines $\mathcal{H}_1, \dots, \mathcal{H}_m$, and updating the target parameters. In the distributed computing system, a direct application of the classical algorithm to implement the optimization in (2) is impossible. Furthermore, in Byzantine setting, there exists a subset of indices $\mathcal{B} \subseteq \{1, \dots, m\}$ representing the set of Byzantine machines. Throughout this paper, we assume that the master machine $\mathcal{H}_0$ can never be corrupted. For $j \in \mathcal{B}$, the information sent from $\mathcal{H}_j$ can be arbitrary or even adversarial. Thus, many popular distributed optimization algorithms such as distributed ADMM (Boyd et al. 2011) and distributed approximate Newton methods (see, *e.g.*, Wang et al. 2017; Jordan et al. 2019; Fan et al. 2019) do not work anymore.

In non-sparse setting, one common way to estimate $\boldsymbol{\beta}^*$ is the distributed gradient descent algorithm for the optimization

$$\operatorname{argmin}_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(Y_i - \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{\beta}), \tag{3}$$

In the t-th iteration of distributed gradient descent algorithm, every local machine $\mathcal{H}_j$ computes a gradient based on the parameters obtained in the $(t-1)$-th iteration $\widehat{\boldsymbol{\beta}}^{(t-1)}$ and its local samples as

$$\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(t-1)}) = -\frac{1}{n} \sum_{i \in \mathcal{H}_j} \boldsymbol{X}_i \mathcal{L}'(Y_i - \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(t-1)}).$$

Then we transmit them to the master machine $\mathcal{H}_0$. In the master machine $\mathcal{H}_0$, we take coordinate-wise sample mean $\bar{\boldsymbol{g}}^{(t-1)}$ among the reported gradients $\{\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(t-1)})\}_{j=0}^{m}$, and update the parameter by the gradient descent $\widehat{\boldsymbol{\beta}}^{(t)} = \widehat{\boldsymbol{\beta}}^{(t-1)} - \eta \bar{\boldsymbol{g}}^{(t-1)}$, where the constant η denotes the step size. However, the sample mean of the gradients is quite sensitive to Byzantine fault. It is not hard to see that the aggregated gradient $\bar{\boldsymbol{g}}^{(t-1)}$ can behave arbitrarily bad even if there is only one Byzantine machine. Therefore, vanilla distributed gradient descent is no longer reliable under the Byzantine setting. To guarantee the robustness of the algorithm, Yin et al. (2018) proposed to use the sample median as a gradient aggregator, namely, we obtain the parameter

$$\widehat{\boldsymbol{\beta}}^{(t)} = \widehat{\boldsymbol{\beta}}^{(t-1)} - \eta \operatorname{med}\{\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(t-1)}) \mid 0 \le j \le m\}, \tag{4}$$

where $\operatorname{med}(\cdot)$ denotes the coordinate-wise median of a set of vectors. It has been shown that the result converges to the true parameters $\boldsymbol{\beta}^*$, even with the presence of a moderate fraction of Byzantine machines.

There are quite a few algorithms which have been developed to hedge against Byzantine failures (see, *e.g.*, Feng et al. 2014; Chen et al. 2017; Blanchard et al. 2017; Xie et al. 2018; Alistarh et al. 2018; Yin et al. 2018, 2019; Su and Xu 2019). The key idea of these works is to apply robust mean estimators so that the gradient information reported from all local machines is aggregated robustly. However, when the true parameter $\boldsymbol{\beta}^*$ has a sparse structure (i.e., the number of non-zero elements s is much smaller than the dimension p), the estimator obtained from the distributed gradient descent algorithm (4) and its variants will no longer possess the optimal convergence rate (see Remark 5 for a more detailed discussion). In fact, there are few works concerning distributed sparse learning in the Byzantine setup.

In this paper, we propose a Byzantine-robust distributed method which can learn the sparse structure for M-estimation. We assume that $\boldsymbol{X}, Y$ are generated from the linear model

$$Y = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}^{*} + \epsilon. \tag{5}$$

For the ease of presentation, the noise ϵ is assumed to be independent with the covariate $\boldsymbol{X}$. To propose a fast Byzantine-robust distributed algorithm for the general loss function, we first develop a novel *square loss transformation* method in this paper. Our strategy is that, under some mild model assumptions, we can construct some pseudo-response $\widetilde{Y}$ in place of the original one, and transform the stochastic optimization problem (1) into the following quadratic optimization

$$\mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\mathbb{E}(\widetilde{Y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})^2.$$

In other words, to approximate the sparse parameter $\boldsymbol{\beta}^{*}$ of M-estimation with a general loss function, we solve the problem of least absolute shrinkage and selection operator (LASSO), as follows

$$\mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\frac{1}{N}\sum_{i=1}^{N}(\widetilde{Y}_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta})^2 + \widetilde{\lambda}_N|\boldsymbol{\beta}|_1, \tag{6}$$

where $\widetilde{Y}_i$ is the empirical version of the constructed pseudo-response, and $\widetilde{\lambda}_N$ is the regularization parameter. As the LASSO problem can be solved by many effective algorithms, this square-loss transformation method greatly facilitates computation. Finally, together with (6) and median-based gradient aggregation, we propose a Square Loss Approximated Robust Distributed (SLARD) method. As we only need to solve a LASSO-type optimization problem at each iteration, our algorithm is much faster than directly solving the original one in (2) even in the non-distributed setting. In our theoretical analysis, we will prove that the SLARD method possesses the near-optimal statistical rate, as well as variable selection consistency. It achieves a fast convergence rate with only a constant number of iterations and hence is both communication-efficient and computationally feasible. The SLARD method applies to a large class of loss functions like square loss and Huber loss in a smooth case and absolute deviation loss in a non-smooth case.

1.1 Paper organization and notations

The remainder of this paper is organized as follows. In Sect. 2, we introduce the idea of square loss transformation and provide the distributed algorithm; Sect. 3 presents some technical assumptions and theoretical results for our proposed methods. The cases of smooth loss and non-smooth loss are discussed separately in Sects. 3.1 and 3.2. Empirical analysis on a synthetic dataset and real-world benchmark dataset are provided in Sect. 4.1 to demonstrate the effectiveness of our method. Concluding remarks are given in Sect. 5. All proofs are deferred to the Appendix.

For every vector $\boldsymbol{v} = (v_1, \dots, v_p)^T$, denote $|\boldsymbol{v}|_2 = \sqrt{\sum_{l=1}^{p} v_l^2}$, $|\boldsymbol{v}|_1 = \sum_{l=1}^{p} |v_l|$, and $|\boldsymbol{v}|_\infty = \max_{1\le l\le p} |v_l|$. Moreover, we use $\mathrm{supp}(\boldsymbol{v}) = \{1 \le l \le p \mid v_l \neq 0\}$ as the support of the vector $\boldsymbol{v}$. For every matrix $\boldsymbol{A} \in \mathbb{R}^{p_1\times p_2}$, define $\|\boldsymbol{A}\| = \sup_{|\boldsymbol{v}|_2=1} |\boldsymbol{A}\boldsymbol{v}|_2$, $\|\boldsymbol{A}\|_{L_2} = \sup_{|\boldsymbol{v}|_2=1} |\boldsymbol{A}\boldsymbol{v}|_\infty$, $\|\boldsymbol{A}\|_\infty = \sup_{|\boldsymbol{v}|_\infty=1} |\boldsymbol{A}\boldsymbol{v}|_\infty$

as various matrix norms, and $\Lambda_{\max}(\boldsymbol{A})$ and $\Lambda_{\min}(\boldsymbol{A})$ as the largest and smallest eigenvalues of $\boldsymbol{A}$ respectively. Moreover, given two subsets of indices, I and J, we use $\boldsymbol{A}_{I\times J}$ to denote the submatrix formed by the rows in I and columns in J. We will use $\mathbb{I}(\cdot)$ as the indicator function. The symbols $\lfloor x\rfloor$ ($\lceil x\rceil$) denote the greatest integer (the smallest integer) not larger than (not less than) x. For two sequences a_n, b_n, we say $a_n \asymp b_n$ when $a_n = O(b_n)$ and $b_n = O(a_n)$ hold at the same time. For simplicity, we denote $\mathbb{S}^{p-1}$ and $\mathbb{B}^p$ as the unit sphere and unit ball in $\mathbb{R}^p$ centered at $\mathbf{0}$. For a given quantile level $0 < \tau < 1$ and a sequence of vectors $\{\boldsymbol{v}_i\}_{i=1}^n \subseteq \mathbb{R}^p$, we denote $\mathrm{Quan}_\tau(\boldsymbol{v}_i \mid 1 \le i \le n)$ as the coordinate-wise τ-th sample quantile of $\{\boldsymbol{v}_i\}_{i=1}^n$. Specifically, we use $\mathrm{med}(\cdot)$ as the coordinate-wise median. Lastly, the generic constants are assumed to be independent of m, n, and p.

2 Proposed methods

In this section, we first introduce a Byzantine-robust algorithm for sparse least square regression. For general loss functions, we propose a *square loss transformation* method. By constructing a pseudo-response, we transform the general M-estimation problem into an ℓ_1-penalized least square regression problem, which becomes much more computationally feasible. Based on this idea, we develop a Byzantine-robust distributed algorithm.

2.1 Byzantine-robust LASSO

Let us start our discussion from the standard distributed LASSO problem. Let $N = (m+1)n$ pairs of i.i.d. observations $\{(\boldsymbol{X}_i, Y_i)\}$ be evenly stored in $m+1$ machines $\{\mathcal{H}_0, \dots, \mathcal{H}_m\}$. Then we want to solve the following LASSO problem

$$\mathrm{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2N}\sum_{i=1}^{N}(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta})^2 + \lambda_N|\boldsymbol{\beta}|_1. \tag{7}$$

However, in a distributed setup, it is communication-costly to solve (7) in a direct way. To reduce the communication burden, an approximate Newton method (Shamir et al. 2014; Wang et al. 2017; Jordan et al. 2019) is always used. In particular, we denote $\boldsymbol{g}_j(\boldsymbol{\beta}) = n^{-1}\sum_{i\in\mathcal{H}_j}(\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta} - Y_i\boldsymbol{X}_i)$ as the local gradient function and $\widehat{\boldsymbol{\Sigma}}_j = n^{-1}\sum_{i\in\mathcal{H}_j}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}$ as the local Hessian on machine $\mathcal{H}_j$ (where $0 \le j \le m$). Given an initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$, every machine $\mathcal{H}_j$ computes the vector $\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(0)})$ and sends it to the master machine $\mathcal{H}_0$. Then a one-step Newton iteration can be informally formulated as follows:

$$\widetilde{\boldsymbol{\beta}}^{(1)} = \widehat{\boldsymbol{\beta}}^{(0)} - (\widehat{\boldsymbol{\Sigma}}_0)^{-1}\frac{1}{m+1}\sum_{j=0}^{m}\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(0)}). \tag{8}$$

For simplicity, here we assume $\widehat{\boldsymbol{\Sigma}}_0$ to be invertible, as we will multiply $\widehat{\boldsymbol{\Sigma}}_0$ on the both sides of (8) in the following discussion. Note that we only approximate Hessian information by observations on the master machine $\mathcal{H}_0$, which significantly reduces communication overhead. Moreover, from (8), $\widetilde{\boldsymbol{\beta}}$ can be equivalently viewed as the solution of the quadratic optimization problem

$$\widetilde{\boldsymbol{\beta}}^{(1)} = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}\widehat{\boldsymbol{\Sigma}}_0\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\left\{\widehat{\boldsymbol{\Sigma}}_0\widehat{\boldsymbol{\beta}}^{(0)} - \frac{1}{m+1}\sum_{j=0}^{m}\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(0)})\right\}. \tag{9}$$

To further encourage sparsity, we can add an ℓ_1-penalty to (9). Substituting that $\widehat{\boldsymbol{\Sigma}}_0 = n^{-1}\sum_{i\in\mathcal{H}_0}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}$, we only need to solve the following regularized optimization problem on the master machine $\mathcal{H}_0$

$$\operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2n}\sum_{i\in\mathcal{H}_0}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\left\{\frac{1}{n}\sum_{i\in\mathcal{H}_0}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)} - \frac{1}{m+1}\sum_{j=0}^{m}\boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(0)})\right\} + \lambda_1|\boldsymbol{\beta}|_1. \tag{10}$$

However, we assume that a subset $\mathcal{B}$ of worker machines are Byzantine, which may send arbitrarily erroneous gradient information to the master machine. In particular, we denote $\boldsymbol{g}_j^{(0)}$ as the gradient information sent from worker machine $\mathcal{H}_j$. Note that we only transmit $p-$dimension vectors from worker machines. Then for normal machines (*i.e.*, $j \notin \mathcal{B}$), we have $\boldsymbol{g}_j^{(0)} = \boldsymbol{g}_j(\widehat{\boldsymbol{\beta}}^{(0)})$. While for $j \in \mathcal{B}$, the Byzantine machine $\mathcal{H}_j$ sends arbitrary values $\boldsymbol{g}_j^{(0)} = *$ to the master machine. In this case, taking the average among $\{\boldsymbol{g}_j^{(0)}\}_{j=0}^{m}$ as in (10) is no longer reliable because sample average is highly sensitive to outliers. To address this challenge, we take the coordinate-wise median among the reported vectors $\{\boldsymbol{g}_j^{(0)}\}_{j=0}^{m}$, namely,

$$\widehat{\boldsymbol{g}}^{(0)} = \operatorname{med}(\boldsymbol{g}_j^{(0)} \mid 0 \le j \le m).$$

Then we can solve the following quadratic optimization problem

$$\widehat{\boldsymbol{\beta}}^{(1)} = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2n}\sum_{i\in\mathcal{H}_0}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\left\{\frac{1}{n}\sum_{i\in\mathcal{H}_0}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)} - \widehat{\boldsymbol{g}}^{(0)}\right\} + \lambda_1|\boldsymbol{\beta}|_1. \tag{11}$$

Continuing by updating the parameters $\widehat{\boldsymbol{\beta}}^{(0)}$ and repeating the above procedure, we propose a Byzantine-robust algorithm for sparse least square regression as shown in Algorithm 1. To give a consistent initial estimator, naturally we can obtain $\widehat{\boldsymbol{\beta}}^{(0)}$ by solving a LASSO problem on the master machine $\mathcal{H}_0$

$$\widehat{\boldsymbol{\beta}}^{(0)} = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\frac{1}{2n}\sum_{i\in\mathcal{H}_0}(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta})^2 + \lambda_0|\boldsymbol{\beta}|_1. \tag{12}$$

This estimator is reliable as we assumed that the master machine $\mathcal{H}_0$ cannot be a Byzantine machine.

Algorithm 1 Byzantine Robust Sparse Least Square Regression

```
Input: Data on local machines $\{(\boldsymbol{X}_i, Y_i) \mid i \in \mathcal{H}_j\}$ for $j = 0, \ldots, m$, the number of iterations $T$, the regularization parameters $\lambda_0, \lambda_t$ for $t = 1, \ldots, T$.
1: Compute the initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$ on the master machine $\mathcal{H}_0$ by solving (12).
2: for $t = 1, \ldots, T$ do
3:     Master $\mathcal{H}_0$ distributes $\widehat{\boldsymbol{\beta}}^{(t-1)}$ to each worker machine $\mathcal{H}_j$ for $j = 1, 2, \ldots, m$.
4:     for $j = 0, \ldots, m$ do
5:         The $j$-th worker machine computes
           $$\boldsymbol{g}_j^{(t-1)} = \begin{cases} n^{-1} \sum_{i \in \mathcal{H}_j} (\boldsymbol{X}_i \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(t-1)} - \boldsymbol{X}_i Y_i) & \text{if } j \notin \mathcal{B}, \\ \text{arbitrary values} & \text{if } j \in \mathcal{B}. \end{cases}$$
           Then the $j$-th worker sends $\boldsymbol{g}_j^{(t-1)}$ back to master machine.
6:     end for
7:     Master machine takes coordinate-wise median $\widehat{\boldsymbol{g}}^{(t-1)} = \mathrm{med}(\boldsymbol{g}_j^{(t-1)} \mid 0 \le j \le m)$, and computes the estimator $\widehat{\boldsymbol{\beta}}^{(t)}$ by solving
       $$\widehat{\boldsymbol{\beta}}^{(t)} = \underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\mathrm{argmin}} \ \frac{1}{2n} \sum_{i \in \mathcal{H}_0} \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{X}_i \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}} \Big\{ \frac{1}{n} \sum_{i \in \mathcal{H}_0} \boldsymbol{X}_i \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(t-1)} - \widehat{\boldsymbol{g}}^{(t-1)} \Big\} + \lambda_t |\boldsymbol{\beta}|_1.$$
8: end for
Output: The final estimator $\widehat{\boldsymbol{\beta}}^{(T)}$.
```

Remark 1 Before moving on to the more general class of loss functions, we shall remark on our settings. In this paper, we adopt the master-slave computing architecture. That is, the master machine $\mathcal{H}_0$ is in charge of data updates, and the rest of the worker machines can only communicate with the master machine (except that the Byzantine machines may collude with each other). Moreover, we assume that the master machine $\mathcal{H}_0$ keeps the same amount of data and can never be corrupted. In Xie et al. (2019, 2020), the authors also assumed that the master machine samples a considerable amount of trustful data so that the proposed algorithm can bear a higher fraction of Byzantine workers. This setting is slightly different from classical Byzantine distributed literature (see, *e.g.*, Yin et al. 2018, 2019; Blanchard et al. 2017; Chen et al. 2017; Su and Xu 2019), where the master machine does not hold any data, and the worker machines are equally likely to be corrupted. Their master machine works as a server where the computations can never be corrupted, and thus, it plays a similar role to our master machine.

Remark 2 In the standard master-slave computing architecture, the delay of data transmission is another important feature in practice. More specifically, since local workers are computing gradients and sending them to the master, by the time a computed gradient arrives, it could already be stale. The asynchronous nature of the distributed system brings new challenges in designing learning algorithms. There has been a vast literature concerning distributed asynchronous optimization like Agarwal and Duchi (2012), Mansoori and Wei (2017), Zhou et al. (2018), and Ren et al. (2020). However, it is not obvious how to directly extend our method to the asynchronous setting. On the one hand, since our method needs to aggregate the gradient information from all workers by taking the coordinate-wise median to filter out the small fraction of Byzantine machines, this aggregation rule may no longer be robust when the gradients come asynchronously. On the other hand, after each round of gradient aggregation, we need to update the parameter by solving the ℓ_1-penalized optimization problem (11), which is time-consuming, makeing it not applicable in practice. Therefore, in this paper, we abstract away the asynchrony for simplicity and leave the problem of Byzantine distributed asynchronous sparse learning as an important future direction.

In practice, the noise ϵ can be heavy-tailed or contaminated by outliers. In these cases, ordinary least square regression is not applicable. Many different loss functions $\mathcal{L}(\cdot)$, like absolute deviation loss (Koenker and Hallock 2001; Koenker 2005) and Huber loss (Huber 1973, 2004), are adopted to overcome these problems. In the following discussion, we propose a method, called *square loss transformation*, to transform the general loss functions into least square loss, which greatly facilitates computation.

2.2 Squared loss transformation

In general, our task is to solve the stochastic optimization problem

$$\boldsymbol{\beta}^* = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\mathbb{E}\{\mathcal{L}(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})\}, \tag{13}$$

where $\mathcal{L}(\cdot)$ is a convex loss function. Denote the sub-gradient of $\mathcal{L}(x)$ as $\mathcal{L}'(x)$, which could be non-continuous. Next, we consider the function $h(x) = \mathbb{E}\{\mathcal{L}'(x+\epsilon)\}$, where the expectation is taken over the randomness of error term ϵ in (5). We assume $h(x)$ is differentiable with respect to x and denote the scalar value $H(0) = h'(0)$. Then for every $\boldsymbol{\beta}$, the Hessian matrix of the loss function can be written as

$$\boldsymbol{H}(\boldsymbol{\beta}) = \nabla|_{\boldsymbol{\beta}}\mathbb{E}\big\{\boldsymbol{X}\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})\big\} = \mathbb{E}\big\{\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}h'(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}^* - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})\big\}.$$

Further denote $\boldsymbol{\Sigma} = \mathbb{E}(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}})$. Given an initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$, a one-step Newton iteration takes the following form

$$\begin{aligned}
\widetilde{\boldsymbol{\beta}}^{(1)} &= \widehat{\boldsymbol{\beta}}^{(0)} + \big\{\boldsymbol{H}(\widehat{\boldsymbol{\beta}}^{(0)})\big\}^{-1}\mathbb{E}\big\{\boldsymbol{X}\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)})\big\}\\
&\approx \widehat{\boldsymbol{\beta}}^{(0)} + \big\{H(0)\boldsymbol{\Sigma}\big\}^{-1}\mathbb{E}\big\{\boldsymbol{X}\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)})\big\}\\
&= \boldsymbol{\Sigma}^{-1}\mathbb{E}\Big[\boldsymbol{X}\Big\{\boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)} + \{H(0)\}^{-1}\mathcal{L}'\big(Y - \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}\big)\Big\}\Big].
\end{aligned}$$

Here we ignore the randomness of the initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$ so that the expectation is only taken with respect to the covariate $\boldsymbol{X}$ and the noise ϵ. If we define the pseudo-response

$$\widetilde{Y} := \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)} + \{H(0)\}^{-1}\mathcal{L}'\big(Y - \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}\big),$$

then the original problem (13) can be approximated by the following least square problem

$$\operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\mathbb{E}\left\{(\widetilde{Y}-\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})^2\right\}. \tag{14}$$

We note that $H(0)$ is unknown. Therefore we need to estimate $H(0)$ with $\widehat{H}^{(0)}(0)$ from the observations $\{(\boldsymbol{X}_i, Y_i)\}_{i=1}^N$ and initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$. Then the pseudo-response can be constructed by

$$\widetilde{Y}_i=\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}+(\widehat{H}^{(0)}(0))^{-1}\mathcal{L}'\left(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}\right).$$

The explicit construction of $\widehat{H}^{(0)}(0)$ will be given in detail in the next section. To further adopt (14) into the sparse Byzantine-distributed setting, we only need to repeat the procedure in Sect. 2.1. More specifically, each machine $\mathcal{H}_j$ locally computes

$$\widetilde{\boldsymbol{g}}_j(\widehat{\boldsymbol{\beta}}^{(0)})=\frac{1}{n}\sum_{i\in\mathcal{H}_j}(\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}-\widetilde{Y}_i\boldsymbol{X}_i)=-\frac{1}{n\widehat{H}^{(0)}(0)}\sum_{i\in\mathcal{H}_j}\boldsymbol{X}_i\mathcal{L}'(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}). \tag{15}$$

Normal machines $\mathcal{H}_j$ (where $j\notin\mathcal{B}$) report $\widetilde{\boldsymbol{g}}_j^{(0)}=\widetilde{\boldsymbol{g}}_j(\widehat{\boldsymbol{\beta}}^{(0)})$ to the master $\mathcal{H}_0$, and Byzantine machines $\mathcal{H}_j$ (where $j\in\mathcal{B}$) report arbitrary values $\widetilde{\boldsymbol{g}}_j^{(0)}=*$. Then the master machine aggregates these gradients by taking the coordinate-wise median

$$\widetilde{\boldsymbol{g}}^{(0)}=\operatorname{med}(\widetilde{\boldsymbol{g}}_j^{(0)}\mid 0\le j\le m), \tag{16}$$

and solves the quadratic optimization problem

$$\widehat{\boldsymbol{\beta}}^{(1)}=\operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p}\frac{1}{2n}\sum_{i\in\mathcal{H}_0}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}-\boldsymbol{\beta}^{\mathrm{T}}\left\{\frac{1}{n}\sum_{i\in\mathcal{H}_0}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}-\widetilde{\boldsymbol{g}}^{(0)}\right\}+\lambda_1|\boldsymbol{\beta}|_1. \tag{17}$$

It is worthwhile to note that, at each iteration, every worker machine only needs to transmit a local density estimator $\widehat{H}_j^{(0)}$ and a vector $\widetilde{\boldsymbol{g}}_j^{(0)}$. Therefore the total communication cost is only $O(p)$. By updating the parameter recursively, it is easy to construct an iterative estimator. In particular, let $\widehat{\boldsymbol{\beta}}^{(t-1)}$ be the $(t-1)$-th round estimator, we construct the estimator $\widehat{H}^{(t-1)}(0)$ for $H(0)$ (see Sect. 2.3 for more details). Then we can define

$$\widetilde{Y}_i=\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)}+(\widehat{H}^{(t-1)}(0))^{-1}\mathcal{L}'\left(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)}\right).$$

Each machine computes a gradient

$$\widetilde{\boldsymbol{g}}_j(\widehat{\boldsymbol{\beta}}^{(t-1)})=-\frac{1}{n\widehat{H}^{(t-1)}(0)}\sum_{i\in\mathcal{H}_j}\boldsymbol{X}_i\mathcal{L}'(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)}),$$

and reports the vector $\widetilde{\boldsymbol{g}}_j^{(t-1)}$. The master aggregates the gradient information by $\widetilde{\boldsymbol{g}}^{(t-1)}=\operatorname{med}(\widetilde{\boldsymbol{g}}_j^{(t-1)}\mid 0\le j\le m)$ and solves the following optimization problem

$$\widehat{\boldsymbol{\beta}}^{(t)} = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{2n}\sum_{i\in\mathcal{H}_0} \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{\beta}^{\mathrm{T}}\left\{\frac{1}{n}\sum_{i\in\mathcal{H}_0}\boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)} - \widetilde{\boldsymbol{g}}^{(t-1)}\right\} + \lambda_t|\boldsymbol{\beta}|_1. \tag{18}$$

Algorithm 2 Square Loss Approximated Robust Distributed (SLARD) method

```
Input: Data on local machines $\{(\boldsymbol{X}_i, Y_i) \mid i \in \mathcal{H}_j\}$ for $j = 0, \dots, m$, the number of iterations $T$, the regularization parameter $\lambda_0, \lambda_t$ for $t = 1, \dots, T$. Moreover, we need a kernel function $\mathcal{K}(\cdot)$ and a sequence of bandwidth $h_t$ for $t = 1, \dots, T$ if the loss function $\mathcal{L}(\cdot)$ is non-smooth.
1:  Compute the initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$ on the master machine $\mathcal{H}_0$ by solving (19).
2:  for $t = 1, \dots, T$ do
3:      Master $\mathcal{H}_0$ distributes $\widehat{\boldsymbol{\beta}}^{(t-1)}$ to each local machine $\mathcal{H}_j$ for $j = 1, 2, \dots, m$.
4:      for $j = 0, \dots, m$ do
5:          The $j$-th worker machine computes the local estimator $\widehat{H}_j^{(t-1)}(0)$ according to equation (23), and sends $\widehat{H}_j^{(t-1)}(0)$ back to master machine.
6:      end for
7:      Master machine computes $\widehat{H}^{(t-1)}(0) = \mathrm{med}\{\widehat{H}_j^{(t-1)}(0) \mid 0 \le j \le m\}$, and transmits it to all local machines.
8:      for $j = 0, \dots, m$ do
9:          The $j$-th worker machine computes
            $\widetilde{\boldsymbol{g}}_j^{(t-1)} = \begin{cases} -(n\widehat{H}^{(t-1)}(0))^{-1}\sum_{i\in\mathcal{H}_j}\boldsymbol{X}_i\mathcal{L}'(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)}) & \text{if } j \notin \mathcal{B}, \\ \text{arbitrary values} & \text{if } j \in \mathcal{B}. \end{cases}$
            Then the $j$-th worker sends $\widetilde{\boldsymbol{g}}_j^{(t)}$ back to master machine.
10:     end for
11:     Master machine takes coordinate-wise median $\widetilde{\boldsymbol{g}}^{(t-1)} = \mathrm{med}(\widetilde{\boldsymbol{g}}_j^{(t-1)} \mid 0 \le j \le m)$, and computes the estimator $\widehat{\boldsymbol{\beta}}^{(t)}$ by solving (18).
12: end for
Output: The final estimator $\boldsymbol{\beta}^{(T)}$.
```

Compared with the original optimization problem in (2), the ℓ_1-penalized quadratic optimization problem in (18) is much more computationally feasible. Therefore we call our proposed method the Square Loss Approximated Robust Distributed (SLARD) method. The entire procedure is presented in Algorithm 2. The regularization parameter $\lambda_0, \dots, \lambda_t$ will be specified in Corollary 2 and Theorem 3 in Sect. 3.

For the choice of initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$, a natural candidate is to solve the ℓ_1-penalized M-estimation problem on the master machine $\mathcal{H}_0$, i.e.,

$$\widehat{\boldsymbol{\beta}}^{(0)} = \operatorname{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}\sum_{i\in\mathcal{H}_0} \mathcal{L}(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}) + \lambda_0|\boldsymbol{\beta}|_1. \tag{19}$$

This is always consistent as we assumed in the beginning that $\mathcal{H}_0$ can never be corrupted. One can also adopt a different estimator for $\widehat{\boldsymbol{\beta}}^{(0)}$ so long as it satisfies Assumption C in Section 3.

Remark 3 Suggested by one reviewer, we would like to compare our square loss transformation with the Distributed Least Squares Approximation (DLSA) method proposed in an unpublished paper (Zhu et al. 2019). In the DLSA method, the authors proposed to let each local machine minimize the unpenalized local empirical loss function $\widehat{\boldsymbol{\beta}}_j^{(0)} = \operatorname{argmin}_{\boldsymbol{\beta}} n^{-1} \sum_{i\in\mathcal{H}_j} \mathcal{L}(Y_i - \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta})$. Then each local machine sends the local estimator $\widehat{\boldsymbol{\beta}}_j^{(0)}$ as well as the local covariance information to the server, and solves the quadratic approximated loss function to obtain a refined estimator. There are three major disparities between DLSA and our approach. Firstly, to eliminate the linear term in the quadratic approximation, their local estimator has to be the minimizer of the unpenalized loss function. Therefore it is not applicable for high-dimensional problems. In contrast, our approach allows the initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$ to be any consistent estimator of the true parameters $\boldsymbol{\beta}^*$. Secondly, they require the workers to send the $p \times p$ local covariance matrix $\widehat{\boldsymbol{\Sigma}}_j = \sum_{i\in\mathcal{H}_j} \boldsymbol{X}_i\boldsymbol{X}_i^{\mathrm{T}}$ to the server, which is more communication-costly than ours, as we only need to send the p dimensional gradient information. Lastly, their approach directly uses the local estimators and the covariance information from all local machines to construct the approximated loss function, which makes it hard to be modified into a Byzantine robust method.

2.3 Robust estimate for *H*(0)

Recall that the function $h(x)$ is defined as $\mathbb{E}\{\mathcal{L}'(x+\epsilon)\}$. The target of this section is to estimate the unknown parameter $H(0) = h'(0)$. For the convex loss function $\mathcal{L}(x)$, again we denote its sub-gradient as $\mathcal{L}'(x)$. Note that we allow $\mathcal{L}(x)$ to be non-smooth, so that $\mathcal{L}'(x)$ could be discontinuous. Assume $\mathcal{L}'(x)$ has finitely many distinct discontinuous points $x_1, \dots, x_K$, and it is differentiable outside of these points. Then we can denote the second-order derivative of $\mathcal{L}(x)$ as $\mathcal{L}''(x)$, which can be defined almost everywhere on $\mathbb{R}$. For the sake of completeness, we define $\mathcal{L}''(x_k) = 0$ on the discrete set of the discontinuous points $x_1, \dots, x_K$. Further let the noise ϵ have probability density function $f(\cdot)$. Then we can give an explicit expression for the function $h(x)$ as follows

$$\begin{aligned} h(x) = \mathbb{E}\{\mathcal{L}'(x+\epsilon)\} &= \int_{-\infty}^{\infty} \mathcal{L}'(x+y)f(y)\mathrm{d}y \\ &= \int_{-\infty}^{x_1-x} \mathcal{L}'(x+y)f(y)\mathrm{d}y + \sum_{k=1}^{K-1}\int_{x_k-x}^{x_{k+1}-x} \mathcal{L}'(x+y)f(y)\mathrm{d}y + \int_{x_K-x}^{\infty} \mathcal{L}'(x+y)f(y)\mathrm{d}y. \end{aligned} \tag{20}$$

To take derivative for $h(x)$, note that $\mathcal{L}'(x)$ is differentiable on the interval (x_k, x_{k+1}). Therefore the derivative of every summand in (20) can be given by

$$
\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}x}\int_{x_k-x}^{x_{k+1}-x}\mathcal{L}'(x+y)f(y)\mathrm{d}y\\
&=\int_{x_k-x}^{x_{k+1}-x}\mathcal{L}''(x+y)f(y)\mathrm{d}y+f(x_k-x)\lim_{\widetilde{x}\to x_k^+}\mathcal{L}'(\widetilde{x})-f(x_{k+1}-x)\lim_{\widetilde{x}\to x_{k+1}^-}\mathcal{L}'(\widetilde{x}).
\end{aligned}
$$

For simplicity we assume $x_0=-\infty$ and $x_{K+1}=\infty$. Since $\mathcal{L}'(x)$ is non-continuous at the points x_k $(1\le k\le K)$, we know the left limit $\lim_{\widetilde{x}\to x_k^-}\mathcal{L}'(\widetilde{x})$ and right limit $\lim_{\widetilde{x}\to x_k^+}\mathcal{L}'(\widetilde{x})$ are not equal. Denote the gap at the point x_k as

$$
\Delta_k=\lim_{\widetilde{x}\to x_k^+}\mathcal{L}'(\widetilde{x})-\lim_{\widetilde{x}\to x_k^-}\mathcal{L}'(\widetilde{x}).
$$

Then we have

$$
\begin{aligned}
H(0)&=\sum_{k=0}^{K}\left\{\int_{x_k}^{x_{k+1}}\mathcal{L}''(y)f(y)\mathrm{d}y+f(x_k)\lim_{\widetilde{x}\to x_k^+}\mathcal{L}'(\widetilde{x})-f(x_{k+1})\lim_{\widetilde{x}\to x_{k+1}^-}\mathcal{L}'(\widetilde{x})\right\}\\
&=\mathbb{E}\{\mathcal{L}''(Y-\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}^*)\}+\sum_{k=1}^{K}\Delta_kf(x_k).
\end{aligned}
\tag{21}
$$

From the above expression, to estimate $H(0)$, we have to conduct kernel density estimation for probability density function $f(x)$ at the K points $x_1,\dots,x_K$. Given a kernel function $\mathcal{K}(\cdot)$ and a bandwidth h_1, the probability density $f(x_k)$ can be estimated on each local machine $\mathcal{H}_j$ as follows

$$
\widehat{f}_j^{(0)}(x_k)=\frac{1}{nh_1}\sum_{i\in\mathcal{H}_j}\mathcal{K}\Big(\frac{Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}-x_k}{h_1}\Big).
$$

Therefore, on each local machine $\mathcal{H}_j$, $H(0)$ can be locally estimated by

$$
\widehat{H}_j^{(0)}(0):=\frac{1}{n}\sum_{i\in\mathcal{H}_j}\mathcal{L}''(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)})+\sum_{k=1}^{K}\frac{\Delta_k}{nh_1}\sum_{i\in\mathcal{H}_j}\mathcal{K}\Big(\frac{Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(0)}-x_k}{h_1}\Big).\tag{22}
$$

Then the worker machine $\mathcal{H}_j$ sends the local estimator $\widehat{H}_j^{(0)}(0)$ to the master machine $\mathcal{H}_0$. Note that in the Byzantine-distributed setting, the Byzantine machine $\mathcal{H}_j$ (where $j\in\mathcal{B}$) may send arbitrary values to the master. To avoid the corruption of outliers, we take a median

$$
\widehat{H}^{(0)}(0)=\operatorname{med}\big\{\widehat{H}_j^{(0)}(0)\mid 0\le j\le m\big\},
$$

which estimates $H(0)$ robustly. For a multi-round algorithm, $H(0)$ should be estimated recursively by the newly updated parameter $\widehat{\boldsymbol{\beta}}^{(t)}$. In particular, let $\widehat{\boldsymbol{\beta}}^{(t-1)}$ be the $(t-1)$-th round estimator. On machine $\mathcal{H}_j$, the local estimator is computed as

$$
\widehat{H}_j^{(t-1)}(0):=\frac{1}{n}\sum_{i\in\mathcal{H}_j}\mathcal{L}''(Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)})+\sum_{k=1}^{K}\frac{\Delta_k}{nh_t}\sum_{i\in\mathcal{H}_j}\mathcal{K}\Big(\frac{Y_i-\boldsymbol{X}_i^{\mathrm{T}}\widehat{\boldsymbol{\beta}}^{(t-1)}-x_k}{h_t}\Big),\tag{23}
$$

and sent to the master machine. Then $\mathcal{H}_0$ aggregates the local estimators by

$$\widehat{H}^{(t-1)}(0) = \mathrm{med}\big\{\widehat{H}_j^{(t-1)}(0) \mid 0 \le j \le m\big\}.$$

The choice of bandwidth h_t at each round will be specified in Theorem 3 of Sect. 3. To get a better understanding of the construction of $\widehat{H}_j^{(0)}(0)$, we will provide detailed discussion for the following three commonly adopted loss functions.

Example 1 (Square loss) In classical least square regression, the loss function is $\mathcal{L}(x) = x^2/2$. Clearly we have $\mathcal{L}''(x) \equiv 1$ for all $x \in \mathbb{R}$. Therefore we can directly use $H(0) = 1$ without aggregating the local estimator $\widehat{H}_j^{(0)}(0)$ from each machine. It is not hard to see that, the loss function in (17) is coincident with (11). Therefore, Algorithm 2 is automatically reduced to Algorithm 1.

Example 2 (Huber loss) The Huber loss function is defined as

$$\mathcal{L}(x) = \begin{cases} x^2/2 & \text{for } |x| \le \delta, \\ \delta|x| - \delta^2/2 & \text{otherwise.} \end{cases}$$

where δ is some pre-determined robustification parameter. In this case, the first-order derivative $\mathcal{L}'(x)$ is continuous, and we can compute that $\mathcal{L}''(x) = \mathbb{I}(|x| \le \delta)$. Therefore $H(0) = \mathbb{P}(|\epsilon| \le \delta)$, and from (22) we know that each local estimator $\widehat{H}_j^{(0)}(0)$ can be constructed by

$$\widehat{H}_j^{(0)}(0) = n^{-1} \sum_{i \in \mathcal{H}_j} \mathbb{I}\big(|Y_i - \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(0)}| \le \delta\big).$$

Example 3 (Absolute deviation loss) In median regression problem, the absolute deviation loss function is defined by

$$\mathcal{L}(x) = |x|.$$

Therefore its derivative $\mathcal{L}'(x) = 1 - 2\mathbb{I}(x \le 0)$ is not differentiable at the point $x_1 = 0$. We can easily obtain that $\mathcal{L}''(x) = 0$ (for the discontinuous point $x_1 = 0$, we can just define $\mathcal{L}''(0) = 0$) and $\Delta_1 = 2$. Therefore from (21) we have $H(0) = 2f(0)$. By Eq. (22), we can estimate $H(0)$ by

$$\widehat{H}_j^{(0)}(0) = \frac{2}{nh_1} \sum_{i \in \mathcal{H}_j} \mathcal{K}\Big(\frac{Y_i - \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(0)}}{h_1}\Big),$$

on each local machine $\mathcal{H}_j$, which is coincident with the two times the kernel density estimation of $f(0)$.

Remark 4 Besides using the coordinate-wise median as the robust mean aggregator, there are some other popular robust mean estimators such as trimmed mean (Yin et al. 2018, 2019), Krum (Blanchard et al. 2017), geometric median (Feng et al. 2014; Chen et al. 2017), and iterative filtering (Su and Xu 2019). It can be seen from Algorithm 2 that we can also collect the local estimators $\widehat{H}_j^{(t-1)}(0)$ and the local gradients $\widetilde{\boldsymbol{g}}_j^{(t)}$ using the aforementioned robust aggregators. For the ease of presentations, we only exhibit the theoretical results for the coordinate-wise median aggregators in the following section. In the simulation studies of Section 4.1, we will present the results of our SLARD method based on trimmed mean and Krum. The definitions of these two aggregators are given in Sect. 4.1.

3 Theoretical properties

In this section, we present some theoretical results for our SLARD method. Denote $S = \mathrm{supp}(\boldsymbol{\beta}^*)$ as the support of the true parameter $\boldsymbol{\beta}^*$ and $s = \mathrm{Card}(S)$ as the sparsity level. For ease of presentation, we use the following notation

$$\mathfrak{G}(X_0, \eta) := \mathbb{E}\{X_0^2 \exp(\eta |X_0|)\},$$

where X_0 is a random variable and $\eta > 0$ is some fixed number.

3.1 Theoretical results for smooth loss

Firstly we investigate the convergence and support recovery results for differentiable loss. More specifically, we assume the derivative $\mathcal{L}'(x)$ is continuous at every point $x \in \mathbb{R}$. In this case, kernel density estimation is not needed and $H(0)$ can be locally estimated by

$$\widehat{H}_j^{(0)}(0) = \frac{1}{n} \sum_{i \in \mathcal{H}_j} \mathcal{L}''(Y_i - \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}^{(0)}).$$

Then we need the following technical assumptions.

Assumption A *There exist some constants* $\eta, C_M > 0$ *such that*

$$\sup_{\boldsymbol{v} \in \mathbb{S}^{p-1}} \mathbb{E}\{\exp(\eta |\boldsymbol{X}^T \boldsymbol{v}|^2)\} \le C_M.$$

Assumption B *Denote* $\boldsymbol{\Sigma} = \mathbb{E}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$. *There exist some constants* $0 < \delta_0 < 1$ *and* $\rho > 0$, *such that*

$$\left\| \boldsymbol{\Sigma}_{S^c \times S} \boldsymbol{\Sigma}_{S \times S}^{-1} \right\|_\infty \le 1 - \delta_0,$$

and $\rho \le \Lambda_{\min}(\boldsymbol{\Sigma}) \le \Lambda_{\max}(\boldsymbol{\Sigma}) \le \rho^{-1}$.

Assumption C *The initial estimator* $\widehat{\boldsymbol{\beta}}^{(0)}$ *satisfies* $|\widehat{\boldsymbol{\beta}}^{(0)} - \boldsymbol{\beta}^*|_2 = O_{\mathbb{P}}(r_n)$, *where* $r_n \to 0$. *Furthermore, we assume that* $\mathbb{P}(\mathrm{supp}(\widehat{\boldsymbol{\beta}}^{(0)}) \subseteq S) \to 1$.

Assumption D *The loss function* $\mathcal{L}(\cdot)$ *satisfies* $\mathbb{E}\{\boldsymbol{X}\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}^*)\} = 0$. *Furthermore, there exist constants* $\eta_1, L_1 > 0$ *such that, for every pair of* $(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2)$, *there is*

$$\mathfrak{G}\left\{ \sup_{\boldsymbol{\beta} : |\boldsymbol{\beta} - \boldsymbol{\beta}_1|_2 \le |\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2|_2} |\mathcal{L}''(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_1) - \mathcal{L}''(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})|, \eta_1 \right\} \le L_1 |\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2|_2,$$
$$\mathbb{E}|\mathcal{L}''(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_1) - \mathcal{L}''(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_2)| \le L_1 \mathbb{E}|\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2)|.$$

Assumption S *There exist some constants $L_s > 0$ such that, for any two points $x_1, x_2 \in \mathbb{R}$, there is*

$$|\mathcal{L}'(x_1) - \mathcal{L}'(x_2)| \le L_s |x_1 - x_2|.$$

Moreover, the local sample size n, number of machines m, parameter dimension p, sparsity level s, initial rate r_n and the fraction of Byzantine machines α_n satisfy the following constraints

$$\max\{ms\log n, s^3\log n\} = o(n), \qquad p = O(n^{\gamma_0}),$$
$$r_n = O\big((s\log n/n)^{1/2}\big), \qquad \alpha_n \in [0, 1/2 - \delta_1),$$

for some constants $\gamma_0 > 0$ and $\delta_1 \in (0, 1/2)$.

In Assumption A we require that the covariate $\boldsymbol{X}$ admits sub-Gaussian distribution, which is common in the literature. It is worthwhile noting that, this can be weakened to some polynomial moment condition by applying more delicate techniques. Assumption B is the standard irrepresentable condition to establish support recovery results (see, *e.g.*, Zhao and Yu 2006; Wainwright 2009; Bühlmann and Van De Geer 2011; Hastie et al. 2015). Assumption C requires consistency and support recovery of the initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$. Recall that our $\widehat{\boldsymbol{\beta}}^{(0)}$ is given by solving the ℓ_1-penalized optimization problem (19) on the master machine $\mathcal{H}_0$. It can be shown that these conditions are satisfied. Assumption D assumes the second-order derivative $\mathcal{L}''(\cdot)$ to be ‘continuous’ in a wider sense. This is weaker than the classical Lipschitz continuity condition, as we can see in Example 2. Huber loss has a non-continuous second-order derivative but satisfies this condition (more detailed justification is relegated to the Appendix). In Assumption S, we assume the first-order derivative of the loss function $\mathcal{L}'(\cdot)$ is Lipschitz continuous. Moreover, some rate constraints on the initial estimator and the quantities m, n, p, s, α_n are needed for our theoretical analysis. We note that the constraint on the fraction of Byzantine machines is almost necessary since the median can be ruined when the number of corruptions exceeds $\lceil m/2 \rceil$.

We begin with the convergence result after the first round of communication.

Theorem 1 *Suppose Assumptions A–D and Assumption S hold. Take the regularization parameter in (17) as*

$$\lambda_1 = C_0\Big(\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + r_n\sqrt{\frac{s\log n}{n}}\Big),$$

where C_0 is a sufficiently large constant. Then we have

$$|\widehat{\boldsymbol{\beta}}^{(1)} - \boldsymbol{\beta}^*|_2 = O_{\mathbb{P}}\Big(\frac{\alpha_n\sqrt{s}}{\sqrt{n}} + \sqrt{\frac{s\log n}{mn}} + r_n\sqrt{\frac{s^2\log n}{n}}\Big).$$

This theorem tells us that, with a proper choice of regularization parameter λ_1, the first round of refinement improves the initial rate from r_n to $\max\{\alpha_n\sqrt{s}/\sqrt{n}+\sqrt{s\log n/(mn)}, r_n\sqrt{s^2\log n/n}\}$, as $\sqrt{s^2\log n/n}=o(1)$ by Assumption S. By applying Theorem 1 recursively, we can obtain the following converging rate for the multi-round algorithm.

Corollary 1 *Suppose Assumptions A–D and Assumption S hold. For each round $1\le t_0\le t$, choose the regularization parameter λ_{t_0} in (18) to be*

$$\lambda_{t_0}=C_0\Big(\frac{\alpha_n}{\sqrt{n}}+\sqrt{\frac{\log n}{mn}}+\frac{r_n}{\sqrt{s}}\Big(\frac{s^2\log n}{n}\Big)^{t_0/2}\Big),$$

where C_0 is sufficiently large. Then we have

$$|\hat{\boldsymbol{\beta}}^{(t)}-\boldsymbol{\beta}^*|_2=O_{\mathbb{P}}\Big(\frac{\alpha_n\sqrt{s}}{\sqrt{n}}+\sqrt{\frac{s\log n}{mn}}+r_n\Big(\frac{s^2\log n}{n}\Big)^{t/2}\Big). \tag{24}$$

We can show that the converging rate of our algorithm will be dominated by the first two terms in (24) within constant steps. In particular, from (24), we know the iteration number t satisfies

$$t\ge\frac{\log n+\log m-\log s-\log\log n}{\log n-2\log s-\log\log n}. \tag{25}$$

Together with the constraint $\max\{ms\log n, s^3\log n\}=o(n)$ in Assumption S, we have

$$\frac{\log n+\log m-\log s-\log\log n}{\log n-2\log s-\log\log n}\le 6+c_0,\quad \text{for some constant } c_0>0.$$

Therefore, if the fraction of Byzantine machines satisfies $\alpha_n=O(\sqrt{\log n/m})$, and $t\ge 6+c_0$, our algorithm achieves a near optimal rate $O_{\mathbb{P}}(\sqrt{s\log n/(mn)})$.

Remark 5 We can compare the converging rate obtained in (24) with existing Byzantine robust gradient descent methods like Yin et al. (2018, 2019), Blanchard et al. (2017), Chen et al. (2017), and Su and Xu (2019). The existing Byzantine robust gradient descent methods have not assumed any sparsity structure on the true parameter $\boldsymbol{\beta}^*$. Therefore, from their theory, the convergence rate of their method would be $O_{\mathbb{P}}(\sqrt{p/(mn)})$, where p is the dimension, and mn is the full sample size. However, from standard Lasso theory, when the parameter $\boldsymbol{\beta}^*$ has sparsity level $s\ll p$, the optimal rate would be $O_{\mathbb{P}}(\sqrt{s/(mn)})$, which is much smaller. Thus these methods cannot achieve the optimal rate for the sparse learning problem. To the best of our knowledge, our method is the first Byzantine robust distributed sparse learning algorithm that has a provable nearly optimal statistical rate.

Next we present results on the support recovery of our estimators $\hat{\boldsymbol{\beta}}^{(1)}$ and $\hat{\boldsymbol{\beta}}^{(t)}$. Recall that $S=\mathrm{supp}(\boldsymbol{\beta}^*)$ is the support of $\boldsymbol{\beta}^*$. We firstly consider the support recovery for $\hat{\boldsymbol{\beta}}^{(1)}$.

Theorem 2 *Assume the same assumptions as in Theorem 1, then we have* $\mathrm{supp}(\hat{\boldsymbol{\beta}}^{(1)})\subseteq S$ *with probability tending to one. Moreover, there is*

$$\left|\widehat{\boldsymbol{\beta}}^{(1)} - \boldsymbol{\beta}^*\right|_\infty = O_{\mathbb{P}}\Big(\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty\Big(\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + r_n\sqrt{\frac{s\log n}{n}}\Big)\Big).$$

We can conclude that once the true parameter $\boldsymbol{\beta}^*$ satisfies

$$\min_{l\in S}|\beta_l^*| \geq C\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty\Big(\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + r_n\sqrt{\frac{s\log n}{n}}\Big),$$

for some sufficiently large constant C, the first round estimator $\widehat{\boldsymbol{\beta}}^{(1)}$ has exact support recovery, i.e., $\mathrm{supp}(\widehat{\boldsymbol{\beta}}^{(1)}) = S$, with probability tending to 1. Similarly we have the recovery result for the t-th round estimator $\widehat{\boldsymbol{\beta}}^{(t)}$.

Corollary 2 *Assume the same conditions as in Corollary 1, then we have* $\mathrm{supp}(\widehat{\boldsymbol{\beta}}^{(t)}) \subseteq S$ *with probability tending to one. Moreover, there is*

$$\left|\widehat{\boldsymbol{\beta}}^{(t)} - \boldsymbol{\beta}^*\right|_\infty = O_{\mathbb{P}}\Big(\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty\Big\{\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + \frac{r_n}{\sqrt{s}}\Big(\frac{s^2\log n}{n}\Big)^{t/2}\Big\}\Big).$$

Compared with Theorem 2 above, the ℓ_∞ error becomes smaller as t grows larger. In particular, from (25), when the iteration number $t \geq 6 + c_0$ and the true parameter $\boldsymbol{\beta}^*$ satisfies

$$\min_{l\in S}|\beta_l^*| \geq C\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty\Big(\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}}\Big), \tag{26}$$

for some sufficiently large constant C, the t-th round estimator $\widehat{\boldsymbol{\beta}}^{(t)}$ satisfies $\mathrm{supp}(\widehat{\boldsymbol{\beta}}^{(t)}) = S$ with probability tending to 1. Specifically when the fraction of Byzantine machines satisfies $\alpha_n = O(\sqrt{\log n/m})$, the condition (26) can be reduced to $\min_{l\in S}|\beta_l^*| \geq C\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty\sqrt{\log n/(mn)}$, which is consistent with the "beta-min" condition of the standard LASSO problem in the single machine setting (see Wainwright 2009).

3.2 Theoretical results for non-smooth loss

For the case of non-smooth loss, from the expression of $H(0)$ in equation (21), we need to estimate the probability density function of the noise ϵ. Therefore, in addition to Assumptions A–D, more assumptions on the kernel function $\mathcal{K}(\cdot)$ and the probability density function $f(\cdot)$ are needed.

Assumption E *There exists a constant* $C_d > 0$ *such that the noise* ϵ *has probability density function* $f(\cdot)$ *which satisfies*

$$|f(x)| \leq C_d, \quad |f(x) - f(y)| \leq C_d|x-y|, \quad \text{for any } x, y \in \mathbb{R}.$$

Moreover, there exists a constant C_l *such that* $\min\{f(x_1), \ldots, f(x_K)\} > C_l > 0$.

Assumption F *The kernel function $\mathcal{K}(\cdot)$ satisfies $\int_{-\infty}^{\infty}\mathcal{K}(u)\mathrm{d}u = 1$ and $\mathcal{K}(u) = 0$ for $|u| \geq 1$. Moreover, $\mathcal{K}(\cdot)$ is Lipschitz continuous with parameter C_k, i.e., $|\mathcal{K}(x) - \mathcal{K}(y)| \leq C_k|x-y|$ holds for an arbitrary $x, y \in \mathbb{R}$.*

Assumption NS *There exist constants $\eta_2, L_2 > 0$ such that, for every pair of $(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2)$, there is*

$$\sup_{1\leq l\leq p} \mathfrak{S}\Big\{ \sup_{\beta:|\beta-\beta_1|_2\leq|\beta_1-\beta_2|_2} \big|X_l\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_1) - X_l\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})\big|, \eta_2 \Big\} \leq L_2|\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2|_2,$$
$$\sup_{1\leq l\leq p} \mathbb{E}\big|X_l\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_1) - X_l\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}_2)\big| \leq L_2|\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2|_2.$$

Moreover, the local sample size n, number of machines m, parameter dimension p, sparsity level s, initial rate r_n and fraction of Byzantine machines α_n satisfy the following constraints

$$s^3 m \log n = o(n^{\gamma_1}), \quad p = O(n^{\gamma_0}), \quad r_n = O\big((s\log n/n)^{1/3}\big), \quad \alpha_n \in [0, 1/2 - \delta_1),$$

for some constants $\gamma_0 > 0, \gamma_1 \in (0,1)$ and $\delta_1 \in (0, 1/2)$.

Assumption E assumes the smoothness of the probability density function of the noise ϵ. It is worth noting that this condition is so mild that it allows the noises to admit very heavy-tailed distributions like the Cauchy distribution. Assumption F imposes integrability, compact support and smoothness conditions on the kernel function $\mathcal{K}(\cdot)$. In Assumption NS, we assume a weaker 'continuous' condition on the gradient $\boldsymbol{X}\mathcal{L}'(Y - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta})$. As we can see in Example 3, the non-smooth absolute deviation loss satisfies this condition (the justification is relegated to the Appendix). Moreover, compared with Assumption S in the smooth case, we need more stringent constraints on m, s, n. More specifically, we need a smaller sparsity and number of machines, and a larger local sample size.

To save space, we directly present the multi-round convergence rate as follows. The one-round results will be given in the Appendix.

Theorem 3 *Suppose Assumptions A–F and Assumption NS hold. For each round $1 \leq t_0 \leq t$, choose the bandwidth h_{t_0} in (23) and the regularization parameter λ_{t_0} in (18) to be*

$$h_{t_0} = C_1\Big(\frac{\alpha_n\sqrt{s}}{\sqrt{n}} + \sqrt{\frac{s\log n}{mn}} + r_n^{2^{1-t_0}}\Big(\frac{s^2\log n}{n}\Big)^{1-2^{1-t_0}}\Big),$$
$$\lambda_{t_0} = C_0\Big(\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + \frac{r_n^{2^{-t_0}}}{\sqrt{s}}\Big(\frac{s^2\log n}{n}\Big)^{1-2^{-t_0}}\Big),$$

where C_0 is sufficiently large. Then we have

$$|\widehat{\boldsymbol{\beta}}^{(t)} - \boldsymbol{\beta}^*|_2 = O_{\mathbb{P}}\left(\frac{\alpha_n\sqrt{s}}{\sqrt{n}} + \sqrt{\frac{s\log n}{mn}} + r_n^{2^{-t}}\Big(\frac{s^2\log n}{n}\Big)^{1-2^{-t}}\right). \tag{27}$$

It is not hard to see that, the rate for non-smooth loss is relatively slower than the multi-round converging rate for smooth loss in Corollary 1. To be more specific, we can compare the change of the third term in (24) and (27) after each round of refinement. In the case of smooth loss, the third term is multiplied with a factor $\sqrt{s^2 \log n/n}$, while for non-smooth loss, the refined third term is the geometric mean between $s^2 \log n/n$ and the original term. From the rate constraints $s^3 m \log n = o(n^{\gamma_1})$ in Assumption NS, we can verify that $s^2 \log n/n = o(\sqrt{s \log n/(mn)})$. It can be shown that the converging rate of our algorithm will be dominated by the first two terms in (27) within constant steps. Similar to the derivation of (25), we can obtain that when the iteration number t satisfies

$$2^t \geq \frac{2\log n - 4\log s - 2\log\log n}{\log n - 3\log s - \log\log n - \log m}, \tag{28}$$

or equivalently $t \geq \log_2(2/(1-\gamma_1) + c_0)$ for some constant $c_0 > 0$, the converging rate would be $O_{\mathbb{P}}(\alpha_n \sqrt{s}/\sqrt{n} + \sqrt{s \log n/(mn)})$. Furthermore, if the fraction of Byzantine machines satisfies $\alpha_n = O(\sqrt{\log n/m})$, our algorithm achieves a near optimal rate up to a logarithm factor.

We can compare the converging rate of our SLARD method obtained in Theorem 3 with the results in Chen et al. (2020), which applies similar square loss transformation to non-smooth quantile loss in a non-Byzantine setup. Their converging rate (Theorem 2) is of the same order as (24) of the smooth loss but not the rate in (27) of the non-smooth loss. We believe the inconsistency comes from the nonlinearity of the median aggregator (see Lemma 8 in the Appendix for more details). It would be interesting to investigate other Byzantine-robust methods that achieve a better rate for the non-smooth loss.

Next we present results on support recovery of our t-th round estimator $\widehat{\boldsymbol{\beta}}^{(t)}$ for the non-smooth loss case.

Theorem 4 *Assume the same assumptions as in Theorem 3, then we have* $\mathrm{supp}(\widehat{\boldsymbol{\beta}}^{(t)}) \subseteq S$ *with probability tending to one. Moreover, there is*

$$\left|\widehat{\boldsymbol{\beta}}^{(t)} - \boldsymbol{\beta}^*\right|_\infty = O_{\mathbb{P}}\left(\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty \left\{\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + \frac{r_n^{-2^{-t}}}{\sqrt{s}}\left(\frac{s^2 \log n}{n}\right)^{1-2^{-t}}\right\}\right).$$

When the true parameter $\boldsymbol{\beta}^*$ satisfies

$$\min_{l\in S} |\beta_l^*| \geq C\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty \left\{\frac{\alpha_n}{\sqrt{n}} + \sqrt{\frac{\log n}{mn}} + \frac{r_n^{-2^{-t}}}{\sqrt{s}}\left(\frac{s^2 \log n}{n}\right)^{1-2^{-t}}\right\}, \tag{29}$$

for some sufficiently large constant C, the t-th round estimator $\widehat{\boldsymbol{\beta}}^{(t)}$ has exact support recovery with probability tending to 1. Furthermore, if the fraction of Byzantine machines α_n satisfies $\alpha_n = O(\sqrt{\log n/m})$ and t satisfies (28), *i.e.*, $t \geq \log_2(2/(1-\gamma_1) + c_0)$, the condition (29) can be reduced to $\min_{l\in S} |\beta_l^*| \geq C\left\|\boldsymbol{\Sigma}_{S\times S}^{-1}\right\|_\infty \sqrt{\log n/(mn)}$, which is coincident with the "beta-min" condition of standard LASSO problem in single machine setting.

4 Empirical analysis

In the empirical analysis, we have carried out two classes of experiments. The first part examines our proposed SLARD method on synthetic data and takes an in-depth look at how each factor (e.g., the attack modes, Byzantine fractions, iteration rounds, etc.) influences the algorithm behavior. The latter is an application of the proposed algorithm to the corresponding sparse linear regression task, intended to assess the practical utility in a Byzantine distributed setting.

4.1 Simulation studies on synthetic data set

In this section, we provide simulation studies to demonstrate the performance of our method on least square regression, Huber regression, and median regression respectively.

We consider the following linear model

$$Y_i = \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{\beta}^* + \epsilon_i, \quad i = 1, \dots, N.$$

Inspired by the simulation studies in Fan et al. (2014), we assume the i.i.d. covariate vectors $\boldsymbol{X}_i^{\mathrm{T}} = (X_{i,1}, \dots, X_{i,p})$ $(i = 1, \dots, N)$ are drawn from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. The covariance matrix $\boldsymbol{\Sigma}$ is a $p \times p$ Toeplitz matrix with its (i, j)-th entry $\Sigma_{ij} = 0.5^{|i-j|}$, where $1 \le i,j \le p$. We fix the dimension $p = 500$. Moreover, we set the p-dimensional true coefficient $\boldsymbol{\beta}^*$ as

$$\boldsymbol{\beta}^* = (2, 0, 1.5, 0, 0.8, 0, 0, 1, 0, 1.75, 0, 0, 0.75, 0, 0, 0.3, \mathbf{0}_{p-16}^{\mathrm{T}})^{\mathrm{T}},$$

which means the sparsity level s is fixed as 7. The data is divided into one master machine $\mathcal{H}_0$ and 100 worker machines $\mathcal{H}_1, \dots, \mathcal{H}_{100}$, each local sample size is $n = 200$. Therefore, the entire sample size is $N = 200 \times (100 + 1)$. Note that the data in master machine cannot be corrupted. The initial estimator $\widehat{\boldsymbol{\beta}}^{(0)}$ is given by solving the ℓ_1-penalized optimization problem on the master machine $\mathcal{H}_0$. For the choice of regularization parameter $\lambda_0, \dots, \lambda_T$, motivated by Zou et al. (2007), Wang et al. (2007), we use the following BIC-type selection criterion

$$\mathrm{BIC}_\lambda = \frac{1}{n} \sum_{i \in \mathcal{H}_0} \mathcal{L}(Y_i - \boldsymbol{X}_i^{\mathrm{T}} \widehat{\boldsymbol{\beta}}_\lambda) + \mathrm{df}_\lambda \times \frac{\log n}{n}, \tag{30}$$

where $\widehat{\boldsymbol{\beta}}_\lambda$ denotes the solution of (18) with λ_t replaced by λ, and df_λ is the number of nonzero elements in $\widehat{\boldsymbol{\beta}}_\lambda$.

Methods for comparison. To illustrate the performance of our SLARD method, we first introduce another two robust mean estimators.

- **Coordinate-wise trimmed mean**: Given a set of vectors $\{\boldsymbol{v}_j \mid 1 \le j \le m\}$ (where $\boldsymbol{v}_j = (v_{j,1}, \dots, v_{j,p})^{\mathrm{T}} \in \mathbb{R}^p$) and a threshold level $\gamma \in [0, 1/2)$, we define the coordinate-wise γ-trimmed mean $\widetilde{\boldsymbol{v}} = (\widetilde{v}_1, \dots, \widetilde{v}_p)^{\mathrm{T}}$ as follows. For each coordinate $l \in \{1, \dots, p\}$, denote S_l as the subset of $\{v_{j,l} \mid 1 \leq j \leq m\}$ with the largest and smallest γ fraction of its elements removed. Then we let $\widetilde{v}_l$ be $|S_l|^{-1} \sum_{l \in S_l} v_{j,l}$.

- **Krum**: Given a set of vectors $\{\boldsymbol{v}_j \mid 1 \leq j \leq m\}$ (where $\boldsymbol{v}_j \in \mathbb{R}^p$) and a threshold level $\gamma \in [0, 1/2)$. For each $j \in \{1, \dots, m\}$, we denote S_j as a subset of $\{\boldsymbol{v}_i \mid 1 \leq i \leq m, i \neq j\}$ which precludes γ fraction of elements having the largest Euclidean distance from the vector $\boldsymbol{v}_j$. We further denote $d_j = \sum_{i \in S_j} |\boldsymbol{v}_i - \boldsymbol{v}_j|_2^2$, then the γ-Krum is defined as $\widetilde{\boldsymbol{v}} = \boldsymbol{v}_{j^*}$ where $j^* = \operatorname{argmin}_{1 \leq j \leq m} d_j$.

Then we compare our SLARD method with the following three alternatives:

1. T-SLARD: We run Algorithm 2, where the gradients $\widetilde{\boldsymbol{g}}_j^{(t-1)}$ and estimators $\widehat{H}_j(0)$ are aggregated using the coordinate-wise trimmed mean. For simplicity we assume the true fraction of Byzantine machines α is known to us, so that we can choose the threshold level γ equal to α.
2. K-SLARD: We run Algorithm 2, where the gradients $\widetilde{\boldsymbol{g}}_j^{(t-1)}$ and estimators $\widehat{H}_j(0)$ are aggregated using Krum. Similarly we assume the threshold level γ equals to the true fraction of Byzantine machines α.
3. SLAD: We perform Algorithm 2, except that the local gradients $\widetilde{\boldsymbol{g}}_j^{(t-1)}$ are aggregated via sample mean. As for the aggregation of local estimators $\widehat{H}_j(0)$, simulation studies show that vanilla sample mean leads to extremely unstable output in Byzantine setup. For convenience of comparisons, we assume that the value of $H(0)$ is known in the SLAD method, albeit it is not realistic in practice.

To solve the ℓ_1-penalized optimization problem (18) with these methods, we uniformly adopt the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) proposed in Beck and Teboulle (2009).

Attacking Modes. The corruption mechanism is given by the following three approaches,

1. Gaussian attack: For Byzantine machine $\mathcal{H}_j$ (where $j \in \mathcal{B}$), assume the true gradient is $\widetilde{\boldsymbol{g}}_j^{(t-1)}$. The reported value from $\mathcal{H}_j$ would be independently generated from the multivariate normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_p)$. Similarly, the local estimator $\widehat{H}_j^{(t-1)}(0)$ will be reported as a random value generated from $\mathcal{N}(0, 1)$.
2. Bit-flip attack: For Byzantine machine $\mathcal{H}_j$ (where $j \in \mathcal{B}$), assume the true gradient is $\widetilde{\boldsymbol{g}}_j^{(t-1)}$. Machine $\mathcal{H}_j$ takes its first five coordinates multiplied with -5 and reports to the master machine $\mathcal{H}_0$. As for the local estimator $\widehat{H}_j^{(t-1)}(0)$, the Byzantine machine $\mathcal{H}_j$ just reports $-5\widehat{H}_j^{(t-1)}(0)$.
3. Omniscient attack: For Byzantine machine $\mathcal{H}_j$ (where $j \in \mathcal{B}$), assume the true gradient is $\widetilde{\boldsymbol{g}}_j^{(t-1)}$. Machine $\mathcal{H}_j$ reports $-100\widetilde{\boldsymbol{g}}_j^{(t-1)}$ to the master machine $\mathcal{H}_0$. As for the local estimator $\widehat{H}_j^{(t-1)}(0)$, the Byzantine machine $\mathcal{H}_j$ reports $-100\widehat{H}_j^{(t-1)}(0)$.

4.1.1 Effect of Byzantine fraction

We first compare the performance of our SLARD method with T-SLARD, K-SLARD, and SLAD by varying the fraction of Byzantine machines. Throughout this section, we only present the simulation results of five-step SLARD (the same for T-SLARD, K-SLARD, SLAD). As we will see in Sect. 4.1.2, a five-step iteration is usually enough for these methods to converge to the near-optimal rate. For each experiment, we repeat 500 independent simulations and report the average number of false positives (FP) and false negatives (FN). Moreover, we plot the estimation error in ℓ_2-norm versus the fraction of Byzantine machines.

Results for least square regression. In the least square regression problem, we assume the noise ϵ admits a standard normal distribution, that is, $\epsilon \sim \mathcal{N}(0,1)$. The results are shown in Fig. 1 and Table 1.

As we can see from Fig. 1, under both a Gaussian attack and a bit-flip attack, the ℓ_2 -error of the mean based SLAD method accumulates quickly as the Byzantine fraction α_n increases. Moreover, SLAD diverges under omniscient attack, hence its corresponding performance curve is omitted. In contrast, the performances of the three SLARD methods are relatively stable as α_n varies, which corroborates the robustness of these approaches. Among them, the K-SLARD method behaves significantly worse than the other two. This is reasonable because the Krum aggregator only picks one local gradient vector among all reported gradients, which results in a coarser gradient estimator and further deteriorates the overall performance. Both median-of-mean and trimmed-mean use more gradient information and therefore lead more accurate estimators. From this picture, we can find that the median-of-mean based SLARD method behaves slightly worse than the trimmed-mean based one. This is probably because of the non-linearity of median aggregation (see also the paragraph before Theorem 4). We note that the T-SLARD method requires the choice of the threshold level γ. To ensure the effectiveness of T-SLARD, we utilize the knowledge of the Byzantine fraction α_n and simply set $\gamma = \alpha_n$. However, in the real world, the Byzantine fraction is not available. In Appendix D of the supplementary material, we study more simulation results on T-SLARD under different threshold levels γ. The results show that T-SLARD behaves comparatively to SLARD only when the threshold level γ is not less than the Byzantine fraction α. However, T-SLARD fails to maintain its performance when γ is smaller than α_n, since it includes some outliers. In practice, the MOM-based SLARD is more favorable as it does not require the knowledge of the Byzantine fraction α_n.

Table 1 shows the results of support recovery of these methods. We can find that SLARD and T-SLARD are comparable, which meets the observation in Fig. 1. Both of them can recover the true support correctly under different attack modes and Byzantine fractions. K-SLARD is also robust against Byzantine failures, albeit with a significantly worse performance regarding false positive and false negative. The mean based SLAD approach is susceptible to the attack modes and Byzantine fractions.

Results for Huber regression. In the Huber regression model, we generate the noise ϵ from the mixture of normal distributions $0.9\mathcal{N}(0,1) + 0.1\mathcal{N}(0,100)$. More precisely, with

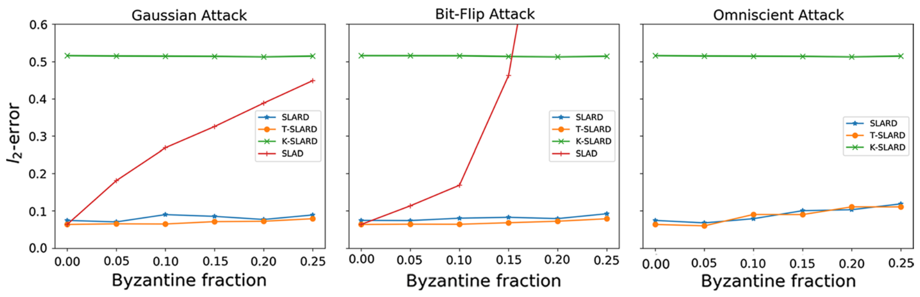

Fig. 1 The ℓ_2-error over Byzantine fraction, under least square regression, varying attack modes. The total sample size N is $200 \times (100+1)$, the number of machines $(m+1)$ is 101, and the dimension p is 500

probability 0.9, the value of ϵ is distributed according to $\mathcal{N}(0,1)$, and is otherwise drawn from a $\mathcal{N}(0,100)$ distribution. For the choice of robustification parameter δ, we follow the classical literature (Huber 2004) and take $\delta = 1.345$. The detailed comparison among these approaches is given in Fig. 2 and Table 2.

From the results, we can observe similar phenomena as in the case of least square regression. The SLARD and T-SLARD perform comparatively in terms of estimation error and support recovery. The Krum based SLARD method behaves worse than them while preserving robustness under different attack modes and Byzantine fractions. The SLAD method has the worst performance because it is not robust to Byzantine failures.

Results for median regression. In the problem of median regression, we generate the noises ϵ from standard Cauchy distribution Cauchy(0, 1). To apply our square loss transformation to the non-smooth loss function $\mathcal{L}(x) = |x|$, we need kernel density estimation (see Example 3). For the choice of kernel function $\mathcal{K}(\cdot)$, we use a biweight kernel function defined as

$$\mathcal{K}(x) = \begin{cases} -\frac{315}{64}x^6 + \frac{735}{64}x^4 - \frac{525}{64}x^2 + \frac{105}{64}, & \text{if } |x| \le 1, \\ 0 & \text{if } |x| > 1. \end{cases}$$

As for the bandwidth h_t, we assume it has form specified in Theorem 3, for the ease of computation, we simply set all the $C_1 = 0.5$ as other choices lead to similar results. The results are shown in Fig. 3 and Table 3. Similar phenomena as the former models can be observed from the simulation results.

4.1.2 Effect of iteration round

In this section, we run experiments with various attack modes and Byzantine fractions and plot how ℓ_2-error changes with rounds of communications under these robust SLARD algorithms. From the previous simulation results, we find that these approaches perform similarly under different models. Due to the page limitation, we only present the result of the Huber regression. Figure 4 summarizes the results, where

Table 1 The false positives (FP) and false negatives (FN) and their standard errors (in parentheses) of the SLARD, T-SLARD, K-SLARD, and SLAD methods under sample size $N = 200 \times (100 + 1)$, local sample size $n = 200$

Attack	None							
α_n	SLARD		T-SLARD		K-SLARD		SLAD	
	FP	FN	FP	FN	FP	FN	FP	FN
0	1.72 (2.52)	0.00 (0.00)	2.19 (5.09)	0.00 (0.00)	4.43 (2.15)	0.12 (0.32)	2.19 (5.09)	0.00 (0.00)
	Gaussian							
0.05	2.64 (1.71)	0.00 (0.00)	1.30 (1.15)	0.00 (0.00)	4.43 (2.13)	0.12 (0.33)	8.15 (3.50)	0.00 (0.00)
0.10	1.04 (0.98)	0.00 (0.00)	2.86 (6.23)	0.00 (0.00)	4.45 (2.13)	0.12 (0.32)	3.83 (2.05)	0.00 (0.00)
0.15	1.59 (1.33)	0.00 (0.00)	2.25 (1.85)	0.00 (0.00)	4.50 (2.13)	0.12 (0.32)	6.05 (2.45)	0.00 (0.04)
0.20	6.50 (2.84)	0.00 (0.00)	3.73 (2.34)	0.00 (0.00)	4.54 (2.14)	0.12 (0.32)	5.46 (2.39)	0.02 (0.13)
0.25	2.79 (1.88)	0.00 (0.00)	3.29 (1.84)	0.00 (0.00)	4.58 (2.12)	0.12 (0.33)	7.71 (2.94)	0.04 (0.21)
	Bit-Flip							
0.05	1.62 (1.33)	0.00 (0.00)	1.12 (1.20)	0.00 (0.00)	4.44 (2.13)	0.12 (0.33)	0.63 (0.71)	0.00 (0.00)
0.10	1.33 (1.12)	0.00 (0.00)	1.63 (1.80)	0.00 (0.00)	4.46 (2.13)	0.12 (0.32)	1.96 (0.94)	0.00 (0.00)
0.15	1.51 (1.15)	0.00 (0.00)	1.65 (1.21)	0.00 (0.00)	4.49 (2.13)	0.12 (0.32)	2.18 (0.94)	0.00 (0.00)
0.20	2.30 (1.41)	0.00 (0.00)	2.23 (1.36)	0.00 (0.00)	4.55 (2.15)	0.12 (0.32)	5.05 (4.44)	0.21 (0.41)
0.25	2.17 (1.37)	0.00 (0.00)	3.01 (1.57)	0.00 (0.00)	4.57 (2.12)	0.12 (0.33)	–	–
	Omniscient							
0.05	4.66 (2.63)	0.00 (0.00)	3.54 (1.89)	0.00 (0.00)	4.43 (2.13)	0.12 (0.33)	–	–
0.10	3.31 (1.80)	0.00 (0.00)	0.84 (0.89)	0.00 (0.00)	4.45 (2.13)	0.12 (0.32)	–	–
0.15	1.37 (1.13)	0.00 (0.00)	1.88 (1.36)	0.00 (0.00)	4.50 (2.13)	0.12 (0.32)	–	–
0.20	3.21 (1.71)	0.00 (0.00)	1.41 (1.18)	0.00 (0.00)	4.54 (2.14)	0.12 (0.32)	–	–
0.25	3.17 (1.69)	0.00 (0.00)	6.48 (2.46)	0.00 (0.00)	4.58 (2.12)	0.12 (0.33)	–	–

The corruption mechanism is given by Gaussian attack, Bit-flip attack, and Omniscient attack. Noises are generated from standard normal distribution $\mathcal{N}(0, 1)$ and the loss function is chosen as square loss

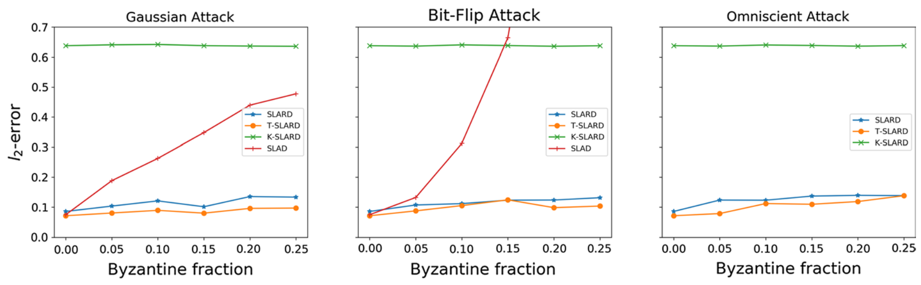

Fig. 2 The ℓ_2-error over Byzantine fraction, under Huber regression, varying attack modes. The total sample size N is $200 \times (100 + 1)$, the number of machines $(m + 1)$ is 101, and the dimension p is 500

the plots are averaged across 100 independent trails. In each plot, we also draw two horizontal lines, representing the ℓ_2-error of solving the ℓ_1-penalized problem using the local data on $\mathcal{H}_0$ (dashed line) and the full data set (solid line), respectively. The corresponding regularization parameters λ are also selected by the BIC-type selection criterion as in (30).

We summarize our observations as follows:

- Both the median-of-mean based and the trimmed-mean based SLARD methods converge very fast. As we can see, under different attack modes and Byzantine fractions, the ℓ_2-errors of these two methods decrease to the global rate within 5-6 rounds of communications, which coincides with our theoretical result (see Corollary 1).
- The ℓ_2-error of SLARD and T-SLARD under omniscient attack is relatively larger than the other two attack modes, which indicates the omniscient attack has a greater impact on the performance of the algorithms.
- The Krum based SLARD method behaves the worst in all settings. More precisely, the ℓ_2-error of K-SLARD only improves a little upon the local rate as the iteration number increases. This is consistent with the phenomenon we have seen in Sect. 4.1.1.

4.1.3 Effect of local sample size

In Corollary 1 of Sect. 3, we have shown that the convergence rate of the SLARD method is of order $O_{\mathbb{P}}(\alpha_n\sqrt{s}/\sqrt{n} + \sqrt{s\log n/(mn)})$, provided the iteration number t is sufficiently large. In this section, we try to corroborate this result through simulation studies. We fix the dimension $p = 500$, the number of machines $m + 1 = 51$, and vary the local sample size n from $\{100, 200, 400, 600, 800\}$. We run experiments for the Huber regression model with various attack modes and Byzantine fractions. Under these constraints, the expected convergence rate should be of order $O_{\mathbb{P}}(n^{-1/2})$. For a better illustration of the relationship, we rescale two axes by logarithm. Figure 5 summarizes the results of five-step SLARD , where the plots are averaged across 100 independent trails. In each plot, we also draw two additional curves, representing the ℓ_2-error of solving the ℓ_1-regularized problem using the

Table 2 The false positives (FP) and false negatives (FN) and their standard errors (in parentheses) of the SLARD, T-SLARD, K-SLARD, and SLAD methods under sample size $N = 200 \times (100 + 1)$, local sample size $n = 200$

Attack	None							
α_n	SLARD		T-SLARD		K-SLARD		SLAD	
	FP	FN	FP	FN	FP	FN	FP	FN
0	1.45 (1.11)	0.00 (0.00)	1.23 (3.94)	0.00 (0.00)	5.33 (2.75)	0.31 (0.46)	0.74 (0.82)	0.00 (0.00)
	Gaussian							
0.05	1.00 (0.97)	0.00 (0.00)	1.03 (0.96)	0.00 (0.00)	4.68 (2.72)	0.32 (0.47)	4.83 (2.34)	0.00 (0.00)
0.10	0.74 (0.85)	0.00 (0.00)	1.46 (3.76)	0.00 (0.00)	5.25 (2.83)	0.32 (0.47)	7.26 (2.74)	0.00 (0.00)
0.15	1.48 (1.17)	0.00 (0.00)	3.16 (2.45)	0.00 (0.00)	5.38 (2.82)	0.32 (0.47)	4.41 (2.15)	0.00 (0.00)
0.20	0.75 (0.84)	0.00 (0.00)	1.75 (1.28)	0.00 (0.00)	5.77 (3.11)	0.31 (0.46)	3.49 (1.94)	0.02 (0.14)
0.25	1.04 (1.04)	0.00 (0.00)	2.61 (1.62)	0.00 (0.00)	5.85 (3.25)	0.32 (0.47)	8.83 (3.05)	0.06 (0.24)
	Bit-Flip							
0.05	0.86 (0.90)	0.00 (0.00)	0.61 (0.71)	0.00 (0.00)	4.97 (2.75)	0.31 (0.46)	0.84 (0.79)	0.00 (0.00)
0.10	0.89 (0.91)	0.00 (0.00)	0.47 (0.62)	0.00 (0.00)	5.35 (2.83)	0.32 (0.47)	1.59 (0.69)	0.00 (0.00)
0.15	0.78 (0.85)	0.00 (0.00)	0.40 (0.59)	0.00 (0.00)	5.38 (2.83)	0.32 (0.47)	2.24 (0.98)	0.00 (0.00)
0.20	1.07 (1.07)	0.00 (0.00)	1.79 (4.79)	0.00 (0.00)	5.77 (3.19)	0.31 (0.46)	6.14 (4.24)	0.26 (0.45)
0.25	1.25 (1.05)	0.00 (0.00)	1.69 (1.52)	0.00 (0.00)	5.57 (3.10)	0.32 (0.47)	34.91 (10.34)	0.42 (0.63)
	Omniscient							
0.05	0.67 (0.79)	0.00 (0.00)	1.58 (1.11)	0.00 (0.00)	4.98 (2.75)	0.31 (0.46)	–	–
0.10	0.90 (0.89)	0.00 (0.00)	0.76 (0.81)	0.00 (0.00)	5.35 (2.83)	0.32 (0.47)	–	–
0.15	0.86 (0.91)	0.00 (0.00)	1.61 (1.27)	0.00 (0.00)	5.39 (2.82)	0.32 (0.47)	–	–
0.20	1.29 (1.16)	0.00 (0.00)	3.91 (1.92)	0.00 (0.00)	5.78 (3.19)	0.31 (0.46)	–	–
0.25	4.32 (2.07)	0.00 (0.00)	4.49 (2.06)	0.00 (0.00)	5.61 (3.11)	0.32 (0.47)	–	–

The corruption mechanism is given by Gaussian attack, Bit-flip attack, and Omniscient attack. Noises are generated from mixture of normal distribution $0.9\mathcal{N}(0, 1) + 0.1\mathcal{N}(0, 100)$ and the loss function is chosen as Huber loss with robustification parameter $\delta = 1.345$

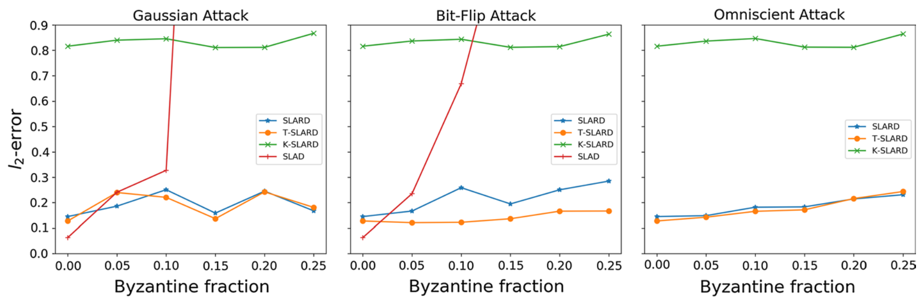

Fig. 3 The ℓ_2-error over Byzantine fraction, under median regression, varying attack modes. The total sample size N is $200 \times (100 + 1)$, the number of machines $(m + 1)$ is 101, and the dimension p is 500

local data on $\mathcal{H}_0$ (dashed line) and the full data set (solid line), respectively. The corresponding regularization parameters λ are also selected by the BIC-type selection criterion as in (30).

From Fig. 5, we can find that the rescaled error curves of the local solution, the global solution, and the SLARD method are nearly parallel to each other, under different attack modes and Byzantine fractions. This indicates that these methods have the nearly same convergence rate to the local sample size n when other quantities are fixed. More precisely, since the slope of these lines is nearly $-1/2$, the rate is close to $O_{\mathbb{P}}(n^{-1/2})$, which is consistent with our theoretical results.

4.2 Application to real-world benchmarks

In this section, to gain some additional insight into algorithm performance, we focus on an application to real-world benchmark data sets.

In the study, we analyze the Ames Housing data set[1], which was compiled by Dean De Cock for use in data science education. This data set contains all sales that had occurred within Ames from 2006 to 2010. The response variable is the house sale price; the covariates include various house features like the lot size, masonry veneer area, and height of the basement. We aim to learn a sparse least square regression model to predict the house price by applying our proposed methods and to compare the performance of different

[1] http://jse.amstat.org/v19n3/decock.pdf.

Table 3 The false positives (FP) and false negatives (FN) and their standard errors (in parentheses) of the SLARD, T-SLARD, K-SLARD, and SLAD methods under sample size $N = 200 \times (100 + 1)$, local sample size $n = 200$

Attack	None							
α_n	SLARD		T-SLARD		K-SLARD		SLAD	
	FP	FN	FP	FN	FP	FN	FP	FN
0	4.90 (2.07)	0.00 (0.00)	3.93 (1.69)	0.00 (0.00)	10.46 (3.08)	0.52 (0.50)	5.36 (1.85)	0.00 (0.00)
	Gaussian							
0.05	1.48 (1.20)	0.00 (0.00)	0.05 (0.22)	0.00 (0.00)	6.26 (2.37)	0.60 (0.49)	2.15 (1.44)	0.00 (0.00)
0.10	0.66 (0.80)	0.00 (0.00)	0.08 (0.31)	0.00 (0.00)	3.60 (1.75)	0.62 (0.49)	2.06 (1.38)	0.00 (0.00)
0.15	4.47 (1.85)	0.00 (0.00)	7.09 (2.22)	0.00 (0.00)	10.57 (3.25)	0.55 (0.50)	63.92 (12.93)	0.00 (0.00)
0.20	0.70 (0.89)	0.00 (0.00)	1.83 (1.35)	0.00 (0.00)	6.41 (2.43)	0.51 (0.50)	5.28 (2.23)	0.03 (0.18)
0.25	2.91 (1.59)	0.00 (0.00)	1.16 (1.10)	0.00 (0.00)	5.50 (2.23)	0.66 (0.48)	95.75 (16.18)	3.87 (2.04)
	Bit-Flip							
0.05	5.33 (2.04)	0.00 (0.00)	5.54 (2.02)	0.00 (0.00)	6.29 (2.34)	0.60 (0.50)	3.21 (1.24)	0.00 (0.00)
0.10	0.18 (0.73)	0.00 (0.04)	4.68 (1.82)	0.00 (0.00)	3.63 (1.77)	0.62 (0.49)	1.73 (0.71)	0.00 (0.00)
0.15	1.71 (1.22)	0.00 (0.00)	3.54 (1.59)	0.00 (0.00)	10.61 (3.23)	0.54 (0.50)	2.04 (0.85)	0.80 (0.40)
0.20	1.30 (1.09)	0.00 (0.00)	6.10 (1.70)	0.00 (0.00)	6.43 (2.47)	0.51 (0.50)	2.57 (0.56)	1.57 (0.61)
0.25	0.20 (0.45)	0.00 (0.00)	5.42 (3.46)	0.00 (0.00)	5.49 (2.20)	0.67 (0.48)	2.76 (0.60)	2.75 (0.46)
	Omniscient							
0.05	5.20 (4.01)	0.00 (0.00)	4.87 (2.05)	0.00 (0.00)	6.32 (2.35)	0.60 (0.50)	–	–
0.10	2.10 (1.37)	0.00 (0.00)	1.73 (1.12)	0.00 (0.00)	3.62 (1.75)	0.62 (0.49)	–	–
0.15	4.73 (1.85)	0.00 (0.00)	5.61 (1.81)	0.00 (0.00)	10.62 (3.27)	0.54 (0.50)	–	–
0.20	3.22 (1.56)	0.00 (0.00)	1.77 (1.21)	0.00 (0.00)	6.42 (2.42)	0.51 (0.50)	–	–
0.25	2.36 (1.44)	0.00 (0.00)	4.50 (1.69)	0.00 (0.00)	5.50 (2.22)	0.67 (0.48)	–	–

The corruption mechanism is given by Gaussian attack, Bit-flip attack, and Omniscient attack. Noises are generated from standard Cauchy distribution Cauchy(0, 1) and the loss function is chosen as absolute deviation loss

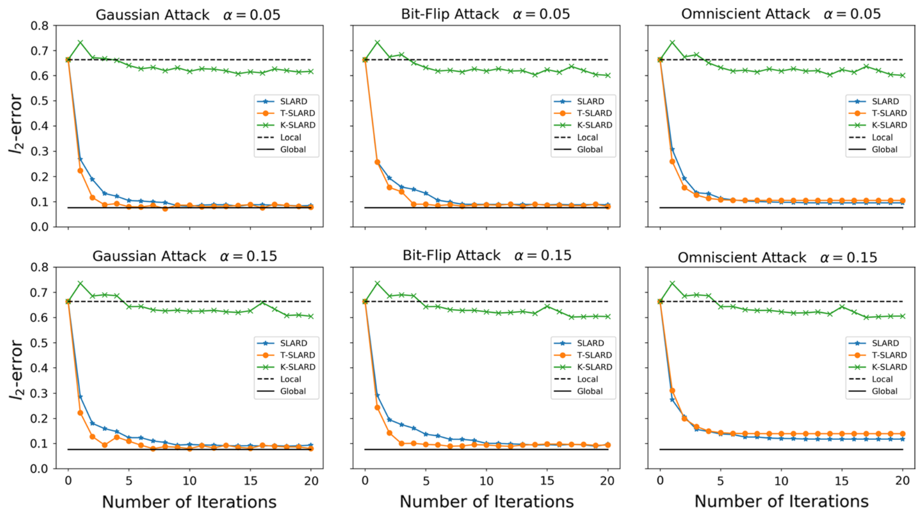

Fig. 4 The ℓ_2-error over iterations, under Huber regression, varying attack modes and Byzantine fractions. The total sample size N is $200 \times (100 + 1)$, the number of machines $(m + 1)$ is 101, and the dimension p is 500

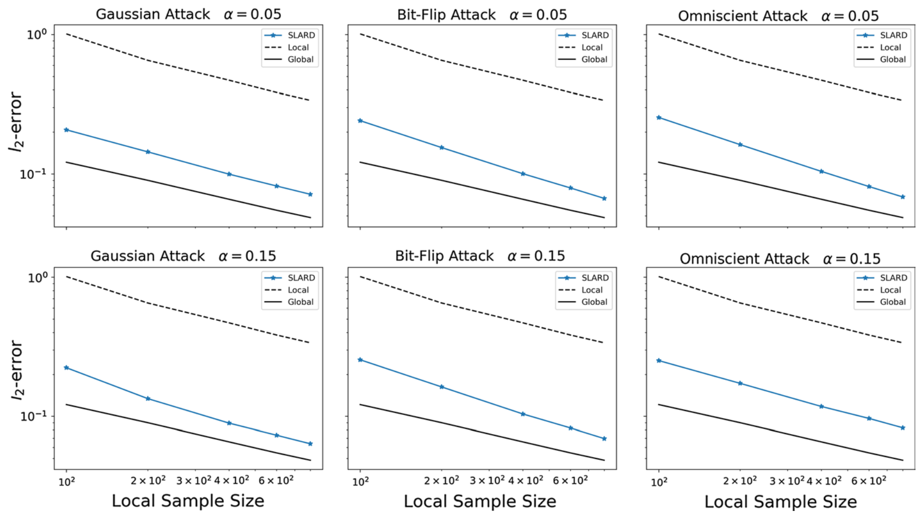

Fig. 5 The ℓ_2-error over local sample sizes, under Huber regression, varying attack modes and Byzantine fractions. The number of machines $(m + 1)$ is 51, and the dimension p is 500

SLARD methods in terms of prediction error. After weeding out the useless features and outliers, we obtained 2902 observations and 70 features. We randomly partitioned the data set into 2100 training data and 802 testing data. To simulate a distributed environment, we further divided the training data into 21 blocks evenly, each holding 100 data points. We chose one block as the master machine $\mathcal{H}_0$, and the rest blocks serve as the worker machines. Among them, the α fraction of machines are assigned as the

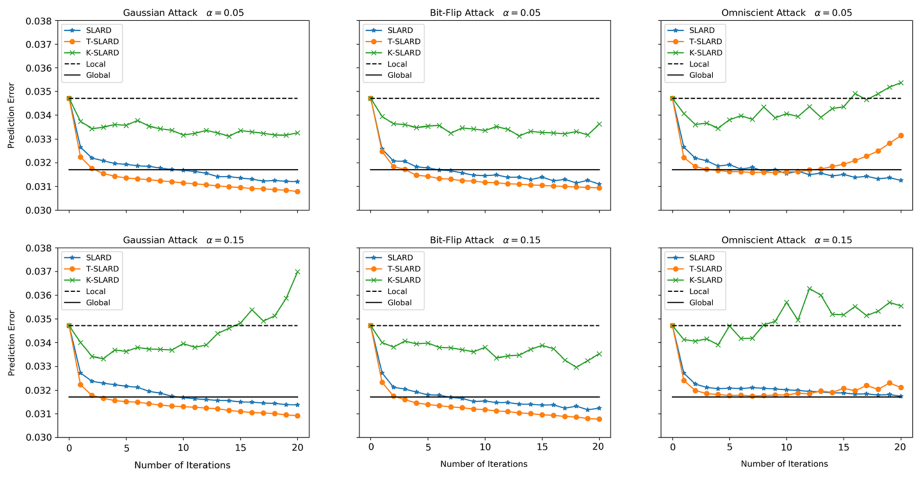

Fig. 6 The prediction error over iterations, under varying attack modes and Byzantine fractions. The total training sample size N is 2100, the test sample size is 802, the number of machines $(m + 1)$ is 21, and the dimension p is 71

Byzantine machines. We vary the fraction α from $\{0.05, 0.15\}$. During the training, each Byzantine machine reports falsified information according to the aforementioned three different attacking modes. For each method, we utilize the BIC-type selection criterion presented in Section 4.1 to choose the regularization parameters. We test 100 random partitions of the training and testing set and report the average prediction error at each iteration on the testing data set. Moreover, we draw two horizontal lines, representing the average prediction error of solving the Lasso problem using the local data on $\mathcal{H}_0$ (dashed line) and the full data set (solid line), respectively. The result is summarized in Fig. 6.

As we can see, the curves of prediction error in Fig. 6 look similar in shape to the curves of ℓ_2-error in Fig. 4. Both SLARD and T-SLARD converge within 5-6 iterations. The trimmed-mean based method seems to be slightly better than the median-of-mean based method. The K-SLARD method behaves the worst among the three. Additionally, it is interesting to find that under gaussian attack and bit-flip attack, the iterative prediction error of SLARD and T-SLARD become smaller than the global prediction error after sufficiently large iteration rounds, which may be due to model misspecification.

5 Concluding remarks

This paper studies the general distributed sparse M-estimation problem with the presence of Byzantine failure. We start from the distributed ℓ_1-penalized least square regression problem and propose to use the coordinate-wise median as a gradient aggregator to hedge against Byzantine corruptions. For general and possibly non-smooth loss functions, we develop a square loss transformation method to convert the target function into the square loss, which greatly alleviates the computational burden. From a theoretical perspective, our method enjoys a fast converging rate and support recovery guarantee. In a future study, as already discussed in Remark 2, we will take into account the delay of data transmission and investigate the Byzantine distributed asynchronous sparse learning problem. Another important future direction is to develop Byzantine-robust algorithms for non-convex loss functions and regularizers. Many kinds of literature (see, *e.g.*, Loh and Wainwright 2015, 2017; Mei et al. 2018; Ma et al. 2019) have shown that non-convexity usually brings about nice practical performance and theoretical properties. It would be interesting to study Byzantine-robust versions of these methods.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10994-021-06001-x.

Acknowledgements Weidong Liu's research is supported by National Program on Key Basic Research Project (2018AAA0100704), NSFC Grant No. 11825104 and 11690013, Youth Talent Support Program, Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). Xiaojun Mao's research is supported by NSFC Grant No. 12001109 and 92046021, Shanghai Sailing Program 19YF1402800, the Science and Technology Commission of Shanghai Municipality grant 20dz1200600. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any funding agency.

References

Agarwal, A., & Duchi, J.C. (2012). Distributed delayed stochastic optimization. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 5451–5452.

Alistarh, D., Allen-Zhu, Z., & Li, J. (2018). Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems*, Curran Associates, Inc., vol 31.

Beck, A., & Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences, 2*(1), 183–202.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R. & Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, Curran Associates, Inc., Vol. 30.

Boyd, S., Parikh, N., Chu, E., Peleato, B., & 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.

Bühlmann, P., & Van De Geer, S. (2011). *Statistics for high-dimensional data: Methods, theory and applications*. Berlin: Springer.

Chen, X., Liu, W., Mao, X., & Yang, Z. (2020). Distributed high-dimensional regression under a quantile loss function. *The Journal of Machine Learning Research, 21*(182), 1–43.

Chen, Y., Su, L., & Xu, J. (2017). Distributed statistical machine learning in adversarial settings. *Proceedings of the ACM on Measurement and Analysis of Computing Systems, 1*(2), 1–25.

Fan, J., Fan, Y., & Barut, E. (2014). Adaptive robust variable selection. *The Annals of Statistics, 42*(1), 324–351.

Fan, J., Guo, Y., & Wang, K. (2019). Communication-efficient accurate statistical estimation. arXiv e-prints arXiv:1906.04870.

Feng, J., Xu, H. & Mannor, S. (2014). Distributed robust learning. arXiv e-prints arXiv:1409.5937.

Hastie, T., Tibshirani, R., & Wainwright, M. (2015). *Statistical learning with sparsity: The Lasso and Generalizations*. Cambridge: CRC Press.

Huber, P. J. (1973). Robust regression: Asymptotics, conjectures and Monte Carlo. *The Annals of Statistics, 1*(5), 799–821.

Huber, P. J. (2004). *Robust statistics* (Vol. 523). New York: Wiley.

Jordan, M. I., Lee, J. D., & Yang, Y. (2019). Communication-efficient distributed statistical inference. *The Journal of the American Statistical Association, 114*(526), 668–681.

Koenker, R. (2005). *Quantile Regression (Econometric Society Monographs; No. 38)*. Cambridge university press

Koenker, R., & Hallock, K. F. (2001). Quantile regression. *Journal of Economic Perspectives, 15*(4), 143–156.

Lamport, L., Shostak, R., & Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems, 4*(3), 382–401.

Lecué, G., & Lerasle, M. (2020). Robust machine learning by median-of-means: Theory and practice. *The Annals of Statistics, 48*(2), 906–931.

Loh, P.-L., & Wainwright, M. J. (2015). Regularized m-estimators with nonconvexity: Statistical and algorithmic theory for local optima. *The Journal of Machine Learning Research, 16*(1), 559–616.

Loh, P.-L., & Wainwright, M. J. (2017). Support recovery without incoherence: A case for nonconvex regularization. *The Annals of Statistics, 45*(6), 2455–2482.

Lugosi, G., & Mendelson, S. (2019). Regularization, sparse recovery, and median-of-means tournaments. *Bernoulli, 25*(3), 2075–2106.

Ma, C., Wang, K., Chi, Y. & Chen, Y. (2019). Implicit regularization in nonconvex statistical estimation: Gradient descent converges linearly for phase retrieval, matrix completion, and blind deconvolution. *Foundations of Computational Mathematics*, 1–182.

Mansoori, F. & Wei, E. (2017). Superlinearly convergent asynchronous distributed network newton method. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2874–2879.

Mei, S., Bai, Yu., & Montanari, A. (2018). The landscape of empirical risk for nonconvex losses. *The Annals of Statistics, 46*(6A), 2747–2774.

Minsker, S. (2015). Geometric median and robust estimation in banach spaces. *Bernoulli, 21*(4), 2308–2335.

Minsker, S. (2019). Distributed statistical estimation and rates of convergence in normal approximation. *The Electronic Journal of Statistics, 13*(2), 5213–5252.

Ren, Z., Zhou, Z., Qiu, L., Deshpande, A., & Kalagnanam, J. (2020). Delay-adaptive distributed stochastic optimization. *Proceedings of the AAAI Conference on Artificial Intelligence, 34*(04), 5503–5510.

Shamir, O., Srebro, N. & Zhang, T. (2014). Communication efficient distributed optimization using an approximate newton-type method. In *Proceedings of the 31st International Conference on Machine Learning*, Vol. 32, pp. 1000–1008.

Su, L., & Xu, J. (2019). Securing distributed gradient descent in high dimensional statistical learning. *Proceedings of the ACM on Measurement and Analysis of Computing Systems3*(1).

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *The Journal of the Royal Statistical Society, Series B (Statistical Methodology), 58*(1), 267–288.

Wainwright, M. J. (2009). Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE Transactions on Information Theory, 55*(5), 2183–2202.

Wang, H., Li, R., & Tsai, C.-L. (2007). Tuning parameter selectors for the smoothly clipped absolute deviation method. *Biometrika, 94*(3), 553–568.

Wang, J., Kolar, M., Srebro, N. & Zhang, T. (2017). Efficient distributed learning with sparsity. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, pp. 3636–3645.

Xie, C., Koyejo, O. & Gupta, I. (2018). Generalized Byzantine-tolerant SGD. arXiv e-prints arXiv:1802.10116.

Xie, C., Koyejo, S. & Gupta, I.. (2019). Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, pp. 6893–6901.

Xie, C., Koyejo, S. & Gupta, I. (2020). Zeno++: Robust fully asynchronous SGD. In *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119, pp. 10495–10503.

Yin, D., Chen, Y., Kannan, R. & Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 5650–5659.

Yin, D., Chen, Y., Kannan, R. & Bartlett, P. (2019). Defending against saddle point attack in Byzantine-robust distributed learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, pp. 7074–7084.

Zhao, P., & Yu, B. (2006). On model selection consistency of lasso. *The Journal of Machine Learning Research, 7*(90), 2541–2563.

Zhou, Z., Mertikopoulos, P., Bambos, N., Glynn, P., Ye, Y., Li, L.-J. & Li, F.-F. (2018). Distributed asynchronous optimization with unbounded delays: How slow can you go?. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 5970–5979.

Zhu, X., Li, F. & Wang, H. (2019). Least squares approximation for a distributed system. arXiv e-prints arXiv:1908.04904.

Zou, H., Hastie, T., & Tibshirani, R. (2007). On the "degrees of freedom'' of the lasso. *The Annals of Statistics, 35*(5), 2173–2192.