BYZANTINE-RESILIENT DISTRIBUTED LARGE-SCALE MATRIX COMPLETION

Feng Lin[*] *Qing Ling*[*] *Zhiwei Xiong*[†]

[*] School of Data and Computer Science, Sun Yat-Sen University
[†] School of Information Science and Technology, University of Science and Technology of China

ABSTRACT

In this paper, we aim at completing a large-scale low-rank matrix over a distributed network, which is subject to Byzantine attacks. We consider solving a nonconvex matrix factorization model with the distributed successive over-relaxation (SOR) method, where the distributed workers compute their private matrices using their own training data and the public matrix sent by the master, while the master updates the public matrix through aggregating the private matrices sent by the workers. However, the Byzantine workers could deliberately send faulty messages to the master so as to bias the optimization process. To address this issue, we propose to replace the aggregation step in the distributed SOR method by several state-of-the-art robust ones: geometric median, median, Krum and h-Krum. We conduct numerical experiments on the Netflix dataset and demonstrate the effectiveness of the proposed robust aggregation strategies in handling Byzantine attacks.

***Index Terms*—** Distributed large-scale optimization, Byzantine attacks, matrix completion

1. INTRODUCTION

The problem of recovering a low-rank or approximately low-rank matrix with missing entries arises in many applications, such as collaborative filtering [1], computer vision [2, 3] and global positioning [4, 5], to name a few. The matrix completion task can be formulated as a convex nuclear norm minimization model [6], or nonconvex rank minimization [7] and matrix factorization [8, 9, 10] models. Popular algorithms include singular value soft thresholding [6], singular value hard thresholding [7], stochastic gradient descent [8, 9], and alternating least squares [10], etc.

Most of the existing algorithms consider solving the matrix completion problem in a single computer or data center, which collects all the observed entries (namely, training data) from their owners. This *centralized learning* scheme, though computationally easy to handle, brings significant privacy risks [11]. The recent affairs such as the iCloud leaks of celebrity photos and the surveillance program of PRISM increase the public concerns on protecting personal information in the big data era. In view of the tradeoff between data utility and user privacy, a novel *federated learning* scheme has been proposed and gained popularity in recent years [12, 13, 14]. Instead of collecting the training data from their owners to a centralized authority, a federated learning system leaves the training data stored locally and distributes the computation across the owners (namely, workers). In addition to addressing the user privacy issue, such a federated learning scheme also utilizes the computation resources of workers. Take the matrix completion task using medical data as an example, every worker representing a hospital holds the training data in its own submatrix and completes the missing entries, with the help of a central master node that fuses the messages sent by all the workers and coordinates the learning process.

However, the federated learning scheme also faces serious security challenges. A worker could be vulnerable to various failures, such as data corruptions, computation errors or even hacker attacks. In these circumstances, the messages sent by the worker to the master are problematic, and hence may lead the federated learning process to a wrong end. In this paper, we consider a general Byzantine failure model [15, 16]. Therein, a Byzantine worker knows all information about all other workers and is able to arbitrarily modify its message sent to the master.

Developing Byzantine-resilient learning algorithms has received a lot of research interest in recent years. Most of the existing works rely on distributed stochastic gradient descent (SGD) as the workhorse, and modify the gradient aggregation rule to cope with the Byzantine attacks. To be specific, in every iteration of SGD, every worker computes a stochastic gradient and sends it to the master, while the master runs a descent step using the mean of the stochastic gradients to update the optimization variable and then sends the variable back to all the workers. When some workers are Byzantine, they can manipulate the messages sent to the master so as to bias the descent direction. To handle the Byzantine attacks, a key observation is that, when the data across the workers are independently and identically distributed (i.i.d.), the stochastic gradients computed at every iteration are i.i.d. too. This fact motivates us to apply robust estimation techniques to aggregate the stochastic gradients. It has been shown that geometric median [17, 18], dimensional median [19, 20] and marginal trimmed mean [21] are Byzantine-resilient in aggregating stochastic gradients comparing to the naive mean. More sophisticated algorithms called as Krum and h-Krum are proposed in [22]. For every candidate stochastic gradient, the master selects a fixed number of nearest neighboring stochastic gradients, computes the distances from the candidate and its neighbors, and defines the average distance as the score of the candidate. Krum selects the candidate with the minimal score as the estimate of the gradient, while h-Krum uses average of h candidates with the minimal scores. Other works include [24] that considers asynchronous Byzantine-resilient SGD and [25] that addresses saddle-point attacks in the nonconvex setting.

In this paper, we consider solving the nonconvex matrix factorization model with the distributed SOR method, where the distributed workers compute their private matrices using their own training data and the public matrix sent by the master, while the master updates the public matrix through aggregating the private matrices sent by the workers. However, the Byzantine workers could deliberately send faulty messages to the master so as to bias the optimization process. To address this issue, we propose to replace the aggregation step in the distributed SOR method by several state-of-the-art robust ones: geometric median, median, Krum and h-Krum. We conduct numerical experiments on the billion-entry Netflix dataset and demonstrate the effectiveness of the proposed robust aggregation strategies in handling Byzantine attacks.

2. PROBLEM FORMULATION AND DISTRIBUTED SOR METHOD

In this paper, we focus on the matrix factorization model for low-rank matrix completion [8,9,10], in the form of

$$\min_{X,Y,Z} \frac{1}{2}\|XY - Z\|_F^2, \ s.t. \ Z_{ij} = W_{ij}, \forall (i,j) \in \Omega. \quad (1)$$

Here $\{W_{ij}, (i,j) \in \Omega\}$ are observed entries of a low-rank matrix $W \in \mathbb{R}^{n\times m}$. We approximate W by $Z \in \mathbb{R}^{n\times m}$ and impose the observation constraints $Z_{ij} = W_{ij}$, for all $(i,j) \in \Omega$. Since Z is low-rank, it can be factorized as the product of two smaller matrices $X \in \mathbb{R}^{n\times r}$ and $Y \in \mathbb{R}^{r\times m}$, where r is an estimate of the rank of Z. By minimizing the cost function $(1/2)\|XY - Z\|_F^2$, we aim to find the best rank-r factorization of Z.

The matrix factorization model (1) can be solved by various algorithms, including stochastic gradient descent [8,9] and alternating least squares [10]. In this paper, we consider the nonlinear successive over-relaxation (SOR) method proposed in [10], which minimizes the cost function with respect to X, Y and Z in an alternating manner. At time t, the updates are

$$Z^t(\omega) = \omega Z^t + (1-\omega)U^t Y^t, \quad (2a)$$

$$X^{t+1} = Z^t(\omega)(Y^t)^T, \quad (2b)$$

$$(U^{t+1}, V^{t+1}) = \text{QR}(X^{t+1}), \quad (2c)$$

$$Y^{t+1} = (U^{t+1})^T Z^t(\omega), \quad (2d)$$

$$Z^{t+1} = U^{t+1}Y^{t+1} + P_\Omega(W - U^{t+1}Y^{t+1}), \quad (2e)$$

where $U^{t+1}V^{t+1}$ is the QR decomposition of X^{t+1} with $U^{t+1} \in \mathbb{R}^{n\times r}$ satisfying $(U^{t+1})^T U^{t+1} = I$ and $V^{t+1} \in \mathbb{R}^{r\times r}$ being upper-diagonal, $\omega \geqslant 1$ is the SOR parameter and $Z^t(\omega) \in \mathbb{R}^{n\times m}$ is an auxiliary SOR matrix. When $\omega = 1$, (2) reduces to the standard alternating least squares algorithm. The weight ω can be fixed, or adaptively adjusted during the optimization process [10].

The SOR method can be conveniently implemented in a distributed way. Suppose that there is a distributed network with 1 master and K workers. We split the matrix W by columns to K submatrices in the form of $W = [W_1, \cdots, W_K]$, where $W_k \in \mathbb{R}^{n\times m_k}$ and $\sum_{k=1}^K m_k = m$. The set Ω is also split into subsets in the form of $\Omega = \Omega_1 \cup \cdots \cup \Omega_K$ accordingly. Likewise, the variables Y, Z and $Z(\omega)$ can be split into $Y = [Y_1, \cdots, Y_K]$ with every $Y_k \in \mathbb{R}^{r\times m_k}$, $Z = [Z_1, \cdots, Z_K]$ with every $Z_k \in \mathbb{R}^{n\times m_k}$ and $Z(\omega) = [Z_1(\omega), \cdots, Z_K(\omega)]$ with every $Z_k(\omega) \in \mathbb{R}^{n\times m_k}$, respectively. Let worker k have the training data in W_k with respect to the set Ω_k, and maintain the updates of the private matrices Y_k and Z_k. On the other hand, the update of the public matrix X is maintained by the master. It is straightforward to rewrite (2) to

$$Z_k^t(\omega) = \omega Z_k^t + (1-\omega)U^t Y_k^t, \quad (3a)$$

$$Q_k^{t+1} = Z_k^t(\omega)(Y_k^t)^T, \quad (3b)$$

$$X^{t+1} = \sum_{k=1}^K Q_k^{t+1}, \quad (3c)$$

$$(U^{t+1}, V^{t+1}) = \text{QR}(X^{t+1}), \quad (3d)$$

$$Y_k^{t+1} = (U^{t+1})^T Z_k^t(\omega), \quad (3e)$$

$$Z_k^{t+1} = U^{t+1}Y_k^{t+1} + P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1}), \quad (3f)$$

The distributed SOR method is described in Algorithm 1. The master first receives Q_k^{t+1} from all workers and sums up Q_k^{t+1} to update $X^{t+1} = \sum_{k=1}^K Q_k^{t+1}$. Then, the master runs QR decomposition $X^{t+1} = U^{t+1}V^{t+1}$, broadcasts U^{t+1} to all workers and receives residuals $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$. The optimization process continues if the overall residual is larger than a predefined threshold, and stops otherwise. For worker k, it first calculates $Z_k^t(\omega) = \omega Z_k^t + (1-\omega)U^t Y_k^t$ and sends $Q_k^{t+1} = Z_k^t(\omega)(Y_k^t)^T$ to the master. After receiving U^{t+1} from the master, it updates $Y_k^{t+1} = (U^{t+1})^T Z_k^t(\omega)$ and $Z_k^{t+1} = U^{t+1}Y_k^{t+1} + P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})$. Finally, the residual $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$ is computed and sent to the master. The algorithm continues until the master stops, and worker k obtains Z_k, which is an approximation of W_k. This distributed scheme fits for the federated learning setting in the sense that the training data and the recovered entries are stored at the workers, while the master only has access to Q_k^{t+1}.

Algorithm 1 Distributed SOR Method

Master

while not convergent **do**
  Receive Q_k^{t+1} from workers;
  Update $X^{t+1} = \sum_{k=1}^K Q_k^{t+1}$;
  Run QR decomposition $X^{t+1} = U^{t+1}V^{t+1}$;
  Broadcast U^{t+1} to all workers;
  Receive residuals $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$ from workers;
  Check the residuals and determine whether to stop or not.
end while

Worker k

Input: U^0, Y_k^0, Z_k^0 and $\{W_{ij}, (i,j) \in \Omega_k\}$;
while not convergent **do**
  Calculate $Z_k^t(\omega) = \omega Z_k^t + (1-\omega)U^t Y_k^t$;
  Send $Q_k^{t+1} = Z_k^t(\omega)(Y_k^t)^T$ to master;
  Receive U^{t+1} from master;
  Update $Y_k^{t+1} = (U^{t+1})^T Z_k^t(\omega)$;
  Update $Z_k^{t+1} = U^{t+1}Y_k^{t+1} + P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})$;
  Send the residual $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$ to master.
end while

3. BYZANTINE-RESILIENT DISTRIBUTED SOR METHOD

In the distributed SOR method, a critical step is that the master aggregates the matrices Q_k^{t+1} received from the workers so as to update the public matrix $X^{t+1} = \sum_{k=1}^K Q_k^{t+1}$. This aggregation step is efficient when all workers are trustworthy, but could be problematic when some workers are Byzantine. As we have discussed in Section 1, the Byzantine workers can send arbitrary Q_k^{t+1} such that the master calculates a wrong X^{t+1}, which shall be sent to all workers and prevent the regular workers from correctly completing their submatrices. We illustrate this scenario in Fig. 1, where f out of K workers are Byzantine and the identities of the Byzantine workers are unknown. Our goal is to develop a robust distributed SOR method that is resilient to Byzantine attacks.

The key idea is to combine Byzantine-resilient aggregation approaches with the distributed SOR method. Observe that the aggregation $X^{t+1} = \sum_{k=1}^K Q_k^{t+1}$ can be represented as

$$X^{t+1} = KMean(Q_1^{t+1}, \cdots, Q_K^{t+1}), \quad (4)$$

where $Mean(Q_1^{t+1}, \cdots, Q_K^{t+1}) = (1/K)\sum_{k=1}^K Q_k^{t+1}$ is the mean of the matrices Q_k^{t+1}. It has been shown in [17, 18, 19, 20, 21, 22] that mean is not robust to Byzantine attacks. Thus, we propose to

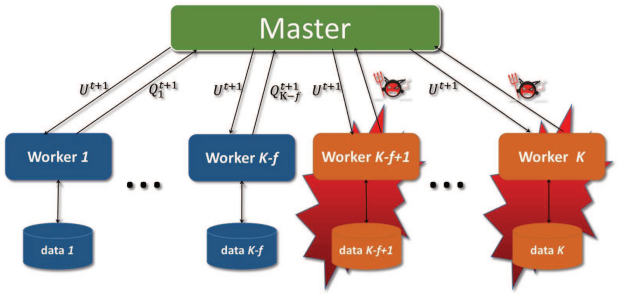

Fig. 1. An illustration of the distributed SOR method under Byzantine attacks. In the master-worker architecture, f out of the total K workers are Byzantine and their identities are unknown.

replace the naive mean by one of the following Byzantine-resilient aggregation operations.

Geometric median. Given a set of matrices $\{Q_k^{t+1}\}$, their geometric median is defined as

$$GeoMed(Q_1^{t+1}, \cdots, Q_K^{t+1}) = \arg\min_{P \in \mathbb{R}^{n\times r}} \sum_{k=1}^{K} \|Q_k^{t+1} - P\|_F,$$

which is unique as long as the matrices Q_k^{t+1} are not collinear. Geometric median provides a centralized trend in multiple dimensions and is widely used in robust statistics [26], as well as Byzantine-robust SGD [17, 18].

A popular approach to calculating the geometric median is the Weiszfeld's algorithm [27, 28] based on iteratively re-weighted least squares. At time τ of the inner loop, it updates

$$P^{\tau+1} = \frac{\sum_{k=1}^{K} Q_k^{t+1}/\|Q_k^{t+1} - P^{\tau}\|_F}{\sum_{k=1}^{K} 1/\|Q_k^{t+1} - P^{\tau}\|_F}.$$

When the initial value of P^0 is properly chosen, the Weiszfeld's algorithm achieves a sublinear rate of convergence of $O(1/\tau)$ [29].

Median. The dimensional median $Median(Q_1^{t+1}, \cdots, Q_K^{t+1})$ calculates the entry-wise median of the submatrices Q_k^{t+1} [19, 20]. It is a computationally cheap substitute for the mean.

Krum. The Krum method returns a robust estimate of the mean, defined as

$$Krum(Q_1^{t+1}, \cdots, Q_K^{t+1}) = Q_{k^*}^{t+1},$$

where

$$k^* = \arg\min_{k\in[K]} \sum_{k\to k'} \|Q_k^{t+1} - Q_{k'}^{t+1}\|_F.$$

For any $k \neq k'$, $k \to k'$ denotes the set containing $K - f - 2$ nearest neighbors of Q_k^{t+1}. Thus, Q_{k*}^{t+1} is the matrix with the minimal summed distance to its nearest neighbors [22]. Note that the Krum method requires that the maximal number of Byzantine workers f is known in advance.

***h*-Krum.** The h-Krum method recursively applies Krum for $h = K - 2f - 2$ rounds and outputs $h\text{-}Krum(Q_1^{t+1}, \cdots, Q_K^{t+1})$ that is the average of the h selected matrices. After every round, the selected matrix Q_k^{t+1} is moved from the candidate set [22]. The same as Krum, it also needs to know the maximal value of f.

With these robust aggregation approaches, we modify the distributed SOR method in Algorithm 1 to a Byzantine-resilient version, as stated in Algorithm 2. The only difference is to replace the aggregation step $X^{t+1} = \sum_{k=1}^{K} Q_k^{t+1}$ by $X^{t+1} = KP_k^{t+1}$ where

$n \times m$	17770×481089
number of training data	99072112
ratio of training data	0.0116
number of test data	1408395

Table 1. Specifications of the Netflix dataset.

P_k^{t+1} is any of the above-mentioned robust estimates for the mean of $\{Q_k^{t+1}\}$. We shall demonstrate the effectiveness of the Byzantine-resilient distributed SOR method with numerical experiments, and compare the performance of different robust aggregation rules.

Algorithm 2 Byzantine-Resilient Distributed SOR Method

Master

while not convergent **do**
  Receive Q_k^{t+1} from all workers;
  Update $X^{t+1} = KP_k^{t+1}$ where P_k^{t+1} is a robust estimate for the mean of $\{Q_k^{t+1}\}$;
  Run QR decomposition $X^{t+1} = U^{t+1}V^{t+1}$;
  Broadcast U^{t+1} to all workers;
  Receive residuals $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$ from workers;
  Check the residuals and determine whether to stop or not.
end while

Regular Worker k

Input: U^0, Y_k^0, Z_k^0 and $\{W_{ij}, (i,j) \in \Omega_k\}$;
while not convergent **do**
  Calculate $Z_k^t(\omega) = \omega Z_k^t + (1-\omega)U^t Y_k^t$;
  Send $Q_k^{t+1} = Z_k^t(\omega)(Y_k^t)^T$ to master;
  Receive U^{t+1} from master;
  Update $Y_k^{t+1} = (U^{t+1})^T Z_k^t(\omega)$;
  Update $Z_k^{t+1} = U^{t+1}Y_k^{t+1} + P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})$;
  Send the residual $\|P_{\Omega_k}(W_k - U^{t+1}Y_k^{t+1})\|_F^2$ to master.
end while

Byzantine Worker k

while not convergent **do**
  Send a manipulated matrix Q_k^{t+1} to master;
  Receive U^{t+1} from master;
  Send a manipulated residual to master.
end while

4. NUMERICAL EXPERIMENTS

In this section, we evaluate the robustness of the proposed Byzantine-resilient distributed SOR method combined with various aggregation rules. We use the full Netflix dataset [23], whose specifications are listed in Table 1. On a computer equipped with 8GB RAM and two Xeon E5-2620 CPUs, we launch 1 master and 20 workers that evenly split the columns of the training data. The algorithms are implemented in C++ and the point-to-point communication is handled by OpenMPI. The recovery performance is evaluated by the root mean square error, defined as

$$RMSE = \sqrt{\sum_{(i,j)\in\mathcal{T}} (\hat{W}_{ij} - W_{ij})^2}.$$

Here $\mathcal{T}$ denotes the test set, W_{ij} is the true value and $\hat{W}_{ij}$ is the corresponding completed value.

In numerical experiments, we consider the proposed Byzantine-resilient distributed SOR method with different aggregation rules:

geometric median, **median**, **Krum** and ***h*-Krum**. We also consider two baseline algorithms: (i) **mean** that is the standard distributed SOR method shown in Algorithm 1, but some workers are Byzantine; and (ii) **mean without Byzantine** where the Byzantine workers are absent. We shall investigate the robustness of these methods under **Guassian attacks** and **sign-flip attacks**. The number of Byzantine workers is chosen as $f = 4$ or $f = 8$. Throughout the numerical experiments, we set the rank estimate $r = 10$, while adaptively adjust the SOR parameter ω according to [10].

With **Gaussian attacks**, at every iteration of the algorithm, every Byzantine worker sends to the master a Gaussian random matrix, whose mean and the standard variation are the same to those of the true one. Observe from Fig. 2 that mean fails in both $f = 4$ and $f = 8$, showing that the standard distributed SOR method is not Byzantine-resilient. When $f = 4$, geometric median and h-Krum are close to mean without Byzantine, and outperform Krum and median. When the number of Byzantine workers is increased to $f = 8$, h-Krum and Krum are better than geometric median, while mean is the worst. Note that both h-Krum and Krum need to exactly know the number of Byzantine workers f. If this knowledge is imperfect, the performance of h-Krum and Krum will deteriorate.

With **sign-flip attacks**, the Byzantine workers multiply every sent entry by a constant $\varepsilon < 0$. In the numerical experiments, we set $\varepsilon = -3$ and depict the results in Fig. 3. In both $f = 4$ and $f = 8$, mean fails and median is the worst among the Byzantine-resilient aggregation rules. Among the other three aggregation rules, geometric median outperforms h-Krum and Krum, demonstrating its robustness to sign-flip attacks.

Without Byzantine attacks, geometric median and h-Krum are able to approximate mean well, as shown in Fig. 4. Median is again the worst among the four Byzantine-resilient aggregation rules. This is reasonable since mean and median could have a remarkable gap. Krum is worse than geometric median and h-Krum, but is much better than median.

In summary, geometric median and h-Krum are robust aggregation rules in all the three cases. When the number of Byzantine attacks is known in advance, h-Krum is a proper choice as its computation complexity is lower than that of geometric median. Otherwise, we recommend to use geometric median, given that the computation overhead is acceptable.

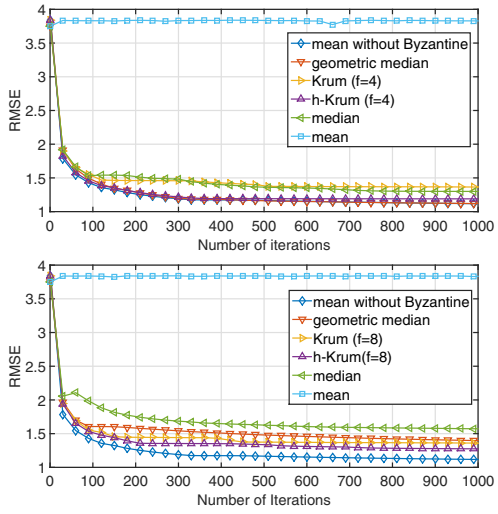

Fig. 2. Gaussian attacks with $f = 4$ (TOP) and $f = 8$ (BOTTOM).

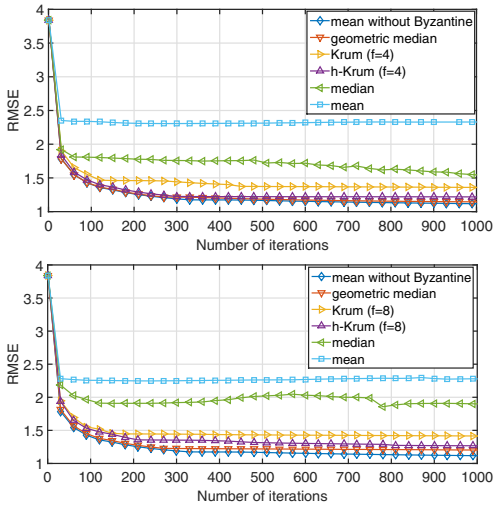

Fig. 3. Sign-flip attacks with $f = 4$ (TOP) and $f = 8$ (BOTTOM).

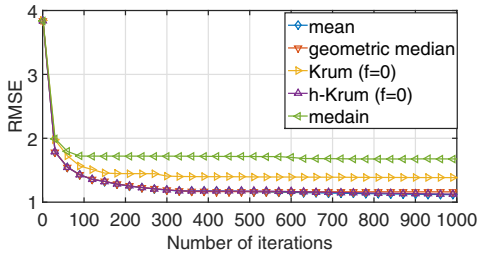

Fig. 4. Without Byzantine attacks.

5. CONCLUSIONS

In this paper, we develop Byzantine-resilient methods to solve the distributed large-scale matrix completion problem. We adopt the distributed SOR method as the optimization framework, and introduce robust aggregation rules for the master to fuse messages received from the workers, among which some might be Byzantine. On the Netflix dataset, we numerically evaluate the performance of four existing robust aggregation rules, including geometric median, mean, Krum and h-Krum. In our future work, we shall further investigate this problem in the following aspects:

1. The robust aggregation rules generally rely on the assumption that the received messages are i.i.d. to identify outliers. This assumption, however, does not necessarily hold in the matrix completion setting. Therefore, statistically establishing the performance guarantee of the robust aggregation rules for matrix completion is of piratical importance.
2. The distributed SOR method requires the master to calculate a QR decomposition at every iteration. To alleviate the computation burden, we shall consider using the light-weight SGD algorithm to solve the matrix factorization model, and leveraging Byzantine-resilient techniques to improve robustness.

Acknowledgement. Qing Ling is supported by NSF China grant 61573331 and NSF Anhui grant 1608085QF130. Zhiwei Xiong is supported by NSF China grant 61671419.

6. REFERENCES

[1] Z. Kang, C. Peng, and Q. Cheng, "Top-N recommender system via matrix completion," In: Proceedings of AAAI, 2016

[2] P. Chen and D. Suter, "Recovering the missing components in a large noisy low-rank matrix: Application to SFM," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 8, pp. 1051–1063, 2004

[3] X. Alamedapineda, E. Ricci, Y. Yan, and N. Sebe, "Recognizing emotions from abstract paintings using non-linear matrix completion," In: Proceedings of CVPR, 2016

[4] A. Montanari and S. Oh, "On positioning via distributed matrix completion," In Proc. of SAM, 2010

[5] J. Cheng, Z. Song, Q. Ye, and H. Du, "MIL: A mobile indoor localization scheme based on matrix completion," In: Proceedings of ICC, 2016

[6] J. Cai, E. Candes, and Z. Shen, "A singular value thresholding algorithm for matrix completion," SIAM Journal on Optimization, vol. 20, no. 4, pp. 1956–1982, 2010

[7] D. Goldfarb and S. Ma, "Convergence of fixed-point continuation algorithms for matrix rank minimization," Foundations of Computational Mathematics, vol. 11, no. 2, pp. 183–210, 2011

[8] R. Gemulla, E. Nijkamp, P. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," In: Proceedings of SIGKDD, 2011

[9] W. Chin, Y. Zhuang, Y. Juan, and C. Lin, "A fast parallel stochastic gradient method for matrix factorization in shared memory systems," ACM Transactions on Intelligent Systems and Technology, vol. 6, no. 1, article 2, 2015

[10] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm," Mathematical Programming Computation, vol. 4, no. 4, pp. 333–361, 2012

[11] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," Computer Networks, vol. 76, pp. 146–164, 2015

[12] J. Konecny, H. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," arXiv: 1511.03575, 2015

[13] J. Konecny, H. McMahan, F. Yu, P. Richtarik, A. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," arXiv: 1610.05492, 2016

[14] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," In: Proceedings of NIPS, 2017

[15] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, pp. 382–401, 1982

[16] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, San Francisco, USA, 1996

[17] Y. Chen, L. Su, and J. Xu. "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," arXiv: 1705.05491, 2017

[18] C. Xie, O. Koyejo, and I. Gupta, "Generalized Byzantine-tolerant SGD," arXiv: 1802.10116, 2018

[19] C. Xie, O. Koyejo, and I. Gupta, "Zeno: Byzantine-suspicious stochastic gradient descent," arXiv: 1805.10032, 2018

[20] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," arXiv: 1803.01498, 2018

[21] C. Xie, O. Koyejo, and I. Gupta, "Phocas: Dimensional Byzantine-resilient stochastic gradient descent," arXiv: 1805.09682, 2018

[22] P. Blanchard, E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," In: Proceedings of NIPS, 2017

[23] J. Bennett and S. Lanning, "The Netflix prize," In Proc. of KDDCup, 2007

[24] G. Damaskinos, E. Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous Byzantine machine learning (the case of SGD)," arXiv: 1802.07928, 2018

[25] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Defending against saddle point attack in Byzantine-robust distributed learning," arXiv: 1806.05358, 2018

[26] H. Cardot, P. Cenac, and P. Zitt, "Efficient and fast estimation of the geometric median in Hilbert spaces with an averaged stochastic gradient algorithm," Bernoulli, vol. 19, no. 1, pp. 18–43, 2013

[27] E. Weiszfeld and F. Plastria, "On the point for which the sum of the distances to n, given points is minimum," Annals of Operations Research, vol. 167, no. 1, pp. 7–41, 2009

[28] U. Eckhardt, "Weber's problem and Weiszfeld's algorithm in general spaces," Mathematical Programming, vol. 18, no. 1, pp. 186–196, 1980

[29] A. Beck and S. Sabach, "Weiszfeld's method: Old and new results," Journal of Optimization Theory applications, vol. 164, no. 1, pp. 1–40, 2015