# Data Encoding Methods for Byzantine-Resilient Distributed Optimization

Deepesh Data
University of California, Los Angeles
CA 90095, USA
Email: deepeshdata@ucla.edu

Linqi Song
City University of Hong Kong
Hong Kong SAR
Email: linqi.song@cityu.edu.hk

Suhas Diggavi
University of California, Los Angeles
CA 90095, USA
Email: suhasdiggavi@ucla.edu

***Abstract*—We consider distributed gradient computation, where both data and computation are distributed among $m$ worker machines, $t$ of which can be Byzantine adversaries, and a designated (master) node computes the model/parameter vector for *generalized linear models*, iteratively, using *proximal gradient descent* (PGD), of which gradient descent (GD) is a special case. The Byzantine adversaries can (collaboratively) deviate arbitrarily from their gradient computation. To solve this, we propose a method based on data encoding and (real) error correction to combat the adversarial behavior. We can tolerate up to $t \leq \lfloor \frac{m-1}{2} \rfloor$ corrupt worker nodes, which is information-theoretically optimal. Our method does not assume any probability distribution on the data. We develop a sparse encoding scheme which enables computationally efficient data encoding. We demonstrate a trade-off between the number of adversaries tolerated and the resource requirement (storage and computational complexity). As an example, our scheme incurs a constant overhead (storage and computational complexity) over that required by the distributed PGD algorithm, without adversaries, for $t \leq \frac{m}{3}$. Our encoding works as efficiently in the streaming data setting as it does in the offline setting, in which all the data is available beforehand.**

## I. Introduction

Map-reduce architecture [1] is implemented in many distributed learning tasks, where there is one designated machine (called the master) that computes the model iteratively, based on the inputs from the worker machines, at each iteration, typically using descent techniques, like (proximal) gradient descent, the Newton's method, etc. The worker nodes perform the required computations using local data, distributed to them [2].

In several applications of distributed learning, including the Internet of Battlefield Things (IoBT) [3], federated optimization [4], the recruited worker nodes might be partially trusted with their computation. Therefore, an important question is whether we can reliably perform distributed computation in the presence of (Byzantine) adversaries, which can arbitrarily deviate from their pre-specified programs. The problem of distributed computation with Byzantine adversaries has a long history [5], and there has been recent interest in applying this model to large-scale distributed learning [6]–[8].

In this paper, we propose a Byzantine-resilient distributed optimization algorithm based on data encoding and error correction (over real numbers). Our proposed algorithm differs from existing Byzantine-resilient distributed learning algorithms in one or more of the following aspects: (i) it does not make statistical assumptions on the data or Byzantine attack patterns; (ii) it is information-theoretically optimal and can tolerate up to a constant fraction ($< 1/2$) of the worker nodes being Byzantine; (iii) it enables a trade-off (in terms of storage and computation overhead in worker nodes) with Byzantine adversary tolerance, without compromising the efficiency at the master node.

Our algorithm encodes the data used by the $m$ worker nodes, using ideas from real-error correction to enable tolerance to Byzantine workers. It develops an efficient "decoding" scheme at the master node, to process the inputs from the workers, to compute the true gradient. It uses a two-phase approach at each iteration of the gradient calculation. The main result (summarized in Theorem 1) demonstrates a trade-off between the Byzantine resilience (in terms of the number of adversarial nodes) and the resource requirement (storage and computational complexity). We can also handle streaming data, where data arrives in batches, rather than being available at the beginning of the computation. Finally, the scheme can handle both Byzantine attacks and missing updates (*e.g.,* caused by delay and asynchrony of worker nodes). Though data encoding is a one-time process, it has to be efficient to harness the advantage of distributed computation. We design a sparse encoding process, based on real-error correction [9], which enables efficient encoding, and the worker nodes alternatively locally encode using the sparse structure. This allows encoding with multiplicative storage redundancy of $2m/(m-2t)$ (which is constant, *even* if $t$ is a constant fraction of $m$), and one-time total computation cost for encoding is $O((1+2t)nd)$, where $n$ is the number of data points each in dimension $d$.

**Paper organization.** Our problem formulation and the main result are stated in Section II, which includes related work. We present our scheme in Section III. Due to space constraints, we give omitted details including an extension of our encoding procedure to the data streaming model and details of our numerical experiments in the full version [10].

**Notation.** We denote vectors by bold small letters (e.g., $\mathbf{x}, \mathbf{y}$, etc.) and matrices by bold capital letters (e.g., $\mathbf{A}, \mathbf{F}$, etc.). Let $|\mathbf{X}|$ denote the total space required to store the matrix $\mathbf{X}$. For any $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. The support of a vector $\mathbf{u} \in \mathbb{R}^n$ is defined by $\mathsf{supp}(\mathbf{u}) := \{i \in [n] : u_i \neq 0\}$.



Authorized licensed use limited to: UCLA Library. Downloaded on August 23,2020 at 06:27:26 UTC from IEEE Xplore. Restrictions apply.

## II. Problem Setting and Our Results

Given a dataset consisting of $n$ data points $\mathbf{x}_i \in \mathbb{R}^d$, $i \in [n]$, we want to learn a *generalized linear model* $\mathbf{w} \in \mathbb{R}^d$, which is a minimizer of the following convex optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) + h(\mathbf{w}) := \left(\frac{1}{n}\sum_{i=1}^{n} \ell(\langle \mathbf{x}_i, \mathbf{w}\rangle)\right) + h(\mathbf{w}), \quad (1)$$

where the loss function $\ell : \mathbb{R} \to \mathbb{R}$ is convex and differentiable, the regularizer $h : \mathbb{R}^d \to \mathbb{R}$ is convex but not necessarily differentiable, and $\langle \mathbf{x}_i, \mathbf{w}\rangle$ is the dot product of $\mathbf{x}_i$ and $\mathbf{w}$. Note that $f(\mathbf{w}) + h(\mathbf{w})$ is a convex function. We can solve (1) using *Proximal Gradient Descent* (PGD) [11]. This is an iterative algorithm, in which we choose an initial $\mathbf{w}_0$ and then update the parameter vector according to the following update rule:

$$\mathbf{w}_{t+1} = prox_{h,\alpha_t}(\mathbf{w}_t - \alpha_t \nabla f(\mathbf{w}_t)), \quad t = 1, 2, 3, \ldots \quad (2)$$

where $\alpha_t$ is the step size or the learning rate at step $t$, determining the convergence behavior. For any $h$ and $\alpha$, the proximal operator $prox_{h,\alpha} : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$prox_{h,\alpha}(\mathbf{w}) = \arg\min_{\mathbf{z}\in\mathbb{R}^d} \frac{1}{2\alpha}\|\mathbf{z} - \mathbf{w}\|_2^2 + h(\mathbf{z}). \quad (3)$$

Observe that if $h = 0$, then $prox_{h,\alpha}(\mathbf{w}) = \mathbf{w}$ for every $\mathbf{w} \in \mathbb{R}^d$, and PGD reduces to the gradient descent (GD). For example *logistic regression*, *linear regression*, etc. If $h(\mathbf{w})$ is convex and differentiable, then we can solve (1) simply using GD. For example, *ridge regression*. Even if $h(\mathbf{w})$ is not convex, it turns out that the corresponding $prox$ operator has a closed form expression for several important optimization problems related to learning. For example, *Lasso*, *SVM dual*, *constrained minimization*, etc. Please see the full version [10] for details.

Let $\mathbf{X} \in \mathbb{R}^{n\times d}$ denote the data matrix, whose $i$'th row is equal to the $i$'th data point $\mathbf{x}_i$. For simplicity, assume that $m$ divides $n$, and let $\mathbf{X}_i$ denote the $\frac{n}{m} \times d$ matrix, whose $j$'th row is equal to $\mathbf{x}_{(i-1)\frac{n}{m}+j}$. In a distributed setup, all the data is distributed among $m$ worker machines (worker $i$ has $\mathbf{X}_i$) and master updates the parameter vector using the update rule (2). At iteration $t$, master sends $\mathbf{w}_t$ to all the workers; worker $i$ computes the gradient (denoted by $\nabla_i f(\mathbf{w}_t)$) on its local data and sends it to the master; master aggregates all the received $m$ local gradients to obtain the global gradient

$$\nabla f(\mathbf{w}_t) = \frac{1}{m}\sum_{i=1}^{m} \nabla_i f(\mathbf{w}_t). \quad (4)$$

Now, master updates the parameter vector according to (2) and obtains $\mathbf{w}_{t+1}$. Repeat the process until convergence. We want to perform this computation under an adversarial attack, where the corrupt worker nodes may provide erroneous vectors. Our adversarial model is described next.

**Adversary model.** In our adversarial model, the adversary can corrupt $t$ of the worker nodes[1], and the compromised nodes may collude and arbitrarily deviate from their prespecified programs. If a worker $i$ is corrupt, then instead of sending the true vector, it may send an arbitrary vector to disrupt the computation. We refer to the corrupt nodes as erroneous or under the Byzantine attack. We can handle asynchronous updates, by dropping the straggling nodes beyond a specified delay, and still compute the correct gradient due to encoding. Therefore we treat updates from these nodes as being "erased". We refer to these as erasures/stragglers. For every worker $i$ that sends a message to the master, we can assume, without loss of generality, that the master receives $\mathbf{u}_i + \mathbf{e}_i$, where $\mathbf{u}_i$ is the true vector, and $\mathbf{e}_i = \mathbf{0}$ if the $i$'th node is honest, otherwise can be arbitrary. We denote the set of nodes under the Byzantine attack by $\mathcal{A}_1$ and straggling nodes by $\mathcal{A}_2$, where $\mathcal{A}_1, \mathcal{A}_2 \subset [m]$, $|\mathcal{A}_1| \le t$, $|\mathcal{A}_2| \le s$, for some $s, t$ that we will decide later. We propose a method that mitigates the effects of both of these anomalies.

**Remark 1.** *A well-studied problem is that of asynchronous distributed optimization, where the workers can have different delays in updates [12]. One mechanism to deal with this is to wait for a subset of responses, before proceeding to the next iteration, treating the others as missing (or erasures) [13]. Byzantine attacks are quite distinct from such erasures, as the adversary can report wrong local gradients, requiring the master node to create mechanisms to overcome such attacks. If the master node simply aggregates the collected updates as in* (4)*, the computed gradient could be arbitrarily far away from the true one, even with a single adversary [14].*

### A. Our Approach

Let $f'(\mathbf{w})$ denote the $n$-length vector whose $j$'th entry is equal to the differentiation of $\ell$ at $\langle \mathbf{x}_j, \mathbf{w}\rangle$, i.e., $(f'(\mathbf{w}))_j = \ell'(u)|_{u=\langle \mathbf{x}_j, \mathbf{w}\rangle}$. With this notation, we can write

$$\nabla f(\mathbf{w}) = \mathbf{X}^T f'(\mathbf{w}), \quad \forall \mathbf{w} \in \mathbb{R}^d. \quad (5)$$

A natural approach for computing the gradient $\nabla f(\mathbf{w})$ is to compute it in two rounds: (i) compute $f'(\mathbf{w})$ in the 1st round by first multiplying $\mathbf{X}$ with $\mathbf{w}$ and then locally obtaining $f'(\mathbf{w})$ from $\mathbf{X}\mathbf{w}$ (we can do this locally, because for every $j \in [n]$, $(\mathbf{X}\mathbf{w})_j = \langle \mathbf{x}_j, \mathbf{w}\rangle$ and $(f'(\mathbf{w}))_j = \ell'(u)|_{u=\langle \mathbf{x}_j, \mathbf{w}\rangle}$); and (ii) compute $\nabla f(\mathbf{w}) = \mathbf{X}^T f'(\mathbf{w})$ in the 2nd round by multiplying $\mathbf{X}^T$ with $f'(\mathbf{w})$. So, the task of each gradient computation reduces to two matrix-vector (MV) multiplications, where the matrices are fixed and vectors may be different each time. To combat against the adversarial worker nodes, we do both of these MV multiplications using data encoding and error correction (over $\mathbb{R}$); see Figure 1 for a pictorial description of our approach. More specifically, for the 1st round, we encode $\mathbf{X}$ using

[1]Our results also apply to a slightly *different* adversarial model, where the adversary can adaptively *choose* which of the $t$ worker nodes to attack at each iteration. However, in this model, the adversary cannot modify the local stored data of the attacked node, as otherwise, over time, it can corrupt all the data, making any defense impossible.



Authorized licensed use limited to: UCLA Library. Downloaded on August 23,2020 at 06:27:26 UTC from IEEE Xplore. Restrictions apply.

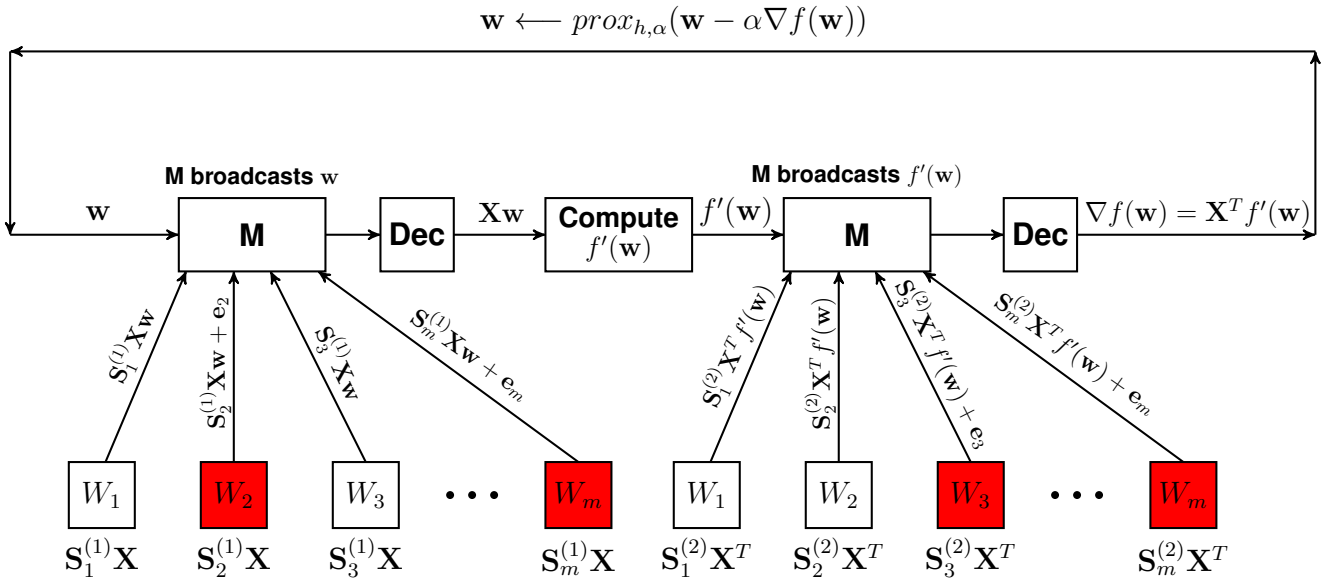


**Fig. 1** This figure shows our 2-round approach to the Byzantine-resilient distributed optimization given in (1). Since gradient is equal to $\nabla f(\mathbf{w}) = \mathbf{X}^T f'(\mathbf{w})$, we compute it in 2 rounds, using the matrix-vector (MV) multiplication as a subroutine in each round. In the 1st round, first we compute $\mathbf{X}\mathbf{w}$, and then compute $f'(\mathbf{w})$ from $\mathbf{X}\mathbf{w}$ – since $(\mathbf{X}\mathbf{w})_j = \langle \mathbf{x}_j, \mathbf{w} \rangle$, we can compute $f'(\mathbf{w})$ from $\mathbf{X}\mathbf{w}$ (see Section II-A). In the 2nd round we compute $\mathbf{X}^T f'(\mathbf{w})$, which is equal to $\nabla f(\mathbf{w})$. For a matrix $\mathbf{A}$ and a vector $\mathbf{v}$, to make our distributed MV multiplication $\mathbf{A}\mathbf{v}$ Byzantine-resilient, we encode $\mathbf{A}$ using a sparse matrix $\mathbf{S} = [\mathbf{S}_1^T\ \mathbf{S}_m^T\ \ldots\ \mathbf{S}_m^T]^T$ and distribute $\mathbf{S}_i\mathbf{A}$ to worker $i$ (denoted by $W_i$). The adversary can corrupt at most $t$ workers (the compromised ones are denoted in red colour), potentially different sets of $t$ workers in different rounds. The master node (denoted by **M**) broadcasts $\mathbf{v}$ to all the workers. Each worker performs the local MV product and sends it back to M. If $W_i$ is corrupt, then it can send an arbitrary vector. Once the master has received all the vectors (out of which $t$ may be erroneous), it sends them to the decoder (denoted by **Dec**), which outputs the correct MV product $\mathbf{A}\mathbf{v}$.

a sparse encoding matrix $\mathbf{S}^{(1)} = [(\mathbf{S}_1^{(1)})^T, \ldots, (\mathbf{S}_m^{(1)})^T]^T$ and store $\mathbf{S}_i^{(1)}\mathbf{X}$ at the $i$'th worker node; and for the 2nd round, we encode $\mathbf{X}^T$ using another sparse encoding matrix $\mathbf{S}^{(2)} = [(\mathbf{S}_1^{(2)})^T, \ldots, (\mathbf{S}_m^{(2)})^T]^T$, and store $\mathbf{S}_i^{(2)}\mathbf{X}^T$ at the $i$'th worker node. Now, in the 1st round of the gradient computation at $\mathbf{w}$, the master node broadcasts $\mathbf{w}$ and the $i$'th worker node replies with $\mathbf{S}_i^{(1)}\mathbf{X}\mathbf{w}$ (a corrupt worker may report an arbitrary vector); upon receiving all the vectors, the master node applies error-correction procedure to recover $\mathbf{X}\mathbf{w}$ and then locally computes $f'(\mathbf{w})$ as described above; in the 2nd round, the master node broadcasts $f'(\mathbf{w})$ and similarly can recover $\mathbf{X}^T f'(\mathbf{w})$ (which is equal to the gradient). Our main result for the Byzantine-resilient distributed gradient computation is as follows:

**Theorem 1** (Main Result)**.** *Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the data matrix. Let $m$ denote the total number of worker nodes. We can compute the gradient exactly in a distributed manner in the presence of $t$ corrupt worker nodes and $s$ stragglers, with the following guarantees, where $\epsilon > 0$ is a free parameter.*

- $(s + t) \leq \left\lfloor \frac{\epsilon}{1+\epsilon} \cdot \frac{m}{2} \right\rfloor$.
- *Total storage requirement is roughly* $2(1 + \epsilon)|\mathbf{X}|$.
- *Computational complexity for each gradient computation:*
  - *At each worker node is* $O((1 + \epsilon)\frac{nd}{m})$.
  - *At the master node is* $O((1 + \epsilon)(n + d)m)$.
- *Total encoding time is* $O\left(nd\left(\frac{\epsilon}{1+\epsilon}m + 1\right)\right)$.

**Remark 2.** *The statement of Theorem 1 allows for any $s$ and $t$, as long as $(s + t) \leq \left\lfloor \frac{\epsilon}{1+\epsilon} \cdot \frac{m}{2} \right\rfloor$. As we are handling both erasures and errors in the same way*[2] *the corruption threshold does not have to handle $s$ and $t$ separately. To simplify the discussion, for the rest of the paper, we consider only Byzantine corruption, and denote the corrupted set by $\mathcal{I} \subset [m]$ with $|\mathcal{I}| \leq t$, with the understanding that this can also work with stragglers.*

**Remark 3.** *Let $m$ be an even number. Note that we can get the corruption threshold $t$ to be any number less than $m/2$, but at the expense of increased storage and computation. For any $\delta > 0$, if we want to get $\delta$ close to m/2, i.e., $t = m/2 - \delta$, then we must have $(1 + \epsilon) \geq m/2\delta$. In particular, at $\epsilon = 2$, we can tolerate up to $m/3$ corrupt nodes, with constant overhead in the total storage as well as on the computational complexity. Our encoding is also efficient and requires $O\left(nd\left(\frac{\epsilon}{1+\epsilon}m + 1\right)\right)$ time. Note that $O(nd)$ is equal to the time required for distributing the data matrix $\mathbf{X}$ among $m$ workers (for running the distributed gradient descent algorithms* without *the adversary); and the encoding time in our scheme (which results in an encoded matrix that provides Byzantine-resiliency) is a factor of $(2t + 1)$ more.*

**Remark 4.** *On comparing the resource requirements of our method with the plain distributed PGD with* no adversarial

[2]When there are *only stragglers*, one can design an encoding scheme where *both* the master and the worker nodes operate oblivious to encoding, while solving a slightly altered optimization problem [13], in which gradients are computed *approximately*, leading to more efficient straggler-tolerant GD.



Authorized licensed use limited to: UCLA Library. Downloaded on August 23,2020 at 06:27:26 UTC from IEEE Xplore. Restrictions apply.

protection[3], *we have that, in our scheme (i) the total storage requirement is $O(1+\epsilon)$ factor more (which is just a constant overhead); (ii) the amount of computation at each worker node is $O(1+\epsilon)$ factor more (which, again, is just a constant overhead); and (iii) the amount of computation at the master node is $O((1+\epsilon)(1+\frac{n}{d}))$ factor more, which is comparable in cases where $n$ is not much bigger than $d$.*

**Remark 5.** *Our scheme is not only efficient (both in terms of computational complexity and storage requirement), but it can also tolerate up to $\lfloor \frac{m-1}{2} \rfloor$ corrupt worker nodes (by taking $\epsilon = m - 1$ in Theorem 1). It is not hard to prove that this bound is information-theoretically optimal, i.e., no algorithm can tolerate $\lceil \frac{m}{2} \rceil$ corrupt worker nodes, and at the same time correctly computes the gradient.*

### B. Related Work

There has been significant recent interest in using coding-theoretic techniques to mitigate the well-known straggler problem [12], including gradient coding [15]–[18], encoding computation [19], [20], data encoding [13]. However, one cannot directly apply the methods for straggler mitigation to the Byzantine attacks case, as we do not know which updates are under attack. Distributed computing with Byzantine adversaries is a richly investigated topic since [5], and has received recent attention in the context of large-scale distributed optimization and learning [6]–[8], [21]. These can be divided into two categories, one which have statistical analysis/assumptions (either explicit statistical models for data [8], [21], or through stochastic GD [6]. Our method gives deterministic guarantees, distinct from these works, but similar in spirit to [7], which is the closest related work.[4] Our storage redundancy *factor* is $2m/(m-2t)$, which is constant, *even* if $t$ is a constant fraction of $m$. In contrast, the storage redundancy *factor* required in [7] is $2t+1$, growing linearly with the number of corrupt worker nodes. This significantly reduces the computation time at the worker nodes in our scheme compared to the scheme in [7], without sacrificing on the computation time required by the master node. Their coding in [7] is restricted to data replication redundancy, as they encode the gradient as done in [15], enabling application to convex problems; in contrast, we encode the data enabling significantly smaller redundancy, and apply it to learn generalized linear models, and is also applicable to MV multiplication. A two-round approach for gradient computation has been proposed for straggler mitigation [19], but our method for MV multiplication differs from that, as we have to provide adversarial protection. Data encoding proposed in [13] applies only to stragglers, and has low-redundancy and complexity, by allowing convergence to an approximate, rather than exact solution.

[3]In plain distributed PGD without any adversarial protection, all the data points are evenly distributed among the $m$ workers. In each iteration, master sends the parameter vector to all the workers; they compute the gradients on their local data in $O(nd/m)$ time (per worker) and send them to the master; master aggregates them in $O(md)$ time to obtain the global gradient and then updates the parameter vector using (2).

[4]In an invited presentation [22], we gave a preliminary version of the results for distributed Byzantine-resilient quadratic unconstrained optimization.

## III. Our Solution

In the section, we give an overview of the core technical part of our two round gradient computation approach – a method of performing distributed matrix-vector (MV) multiplication in the presence of a malicious adversary. Given a fixed matrix $\mathbf{A} \in \mathbb{R}^{n_r \times n_c}$ and a vector $\mathbf{v} \in \mathbb{R}^{n_c}$, we want to compute $\mathbf{Av}$ in a distributed manner in the presence of at most $t$ corrupt worker nodes; see Section II for details on the model. Our method is based on data encoding and real error correction, where the matrix $\mathbf{A}$ is encoded and distributed among all the worker nodes, and the master recovers the MV product $\mathbf{Av}$ using real error correction; see Figure 1.

**A trivial approach.** Take a generator matrix $\mathbf{G}$ of any real error correcting linear code. Encode $\mathbf{A}$ as $\mathbf{A}^T\mathbf{G} =: \mathbf{B}$ and disperse the columns of $\mathbf{B}$ among the worker nodes. Master broadcasts $\mathbf{v}$; responses from the workers are combined as $\mathbf{v}^T\mathbf{B} + \mathbf{e}^T$, where $|\mathsf{supp}(\mathbf{e})| \leq t$. Since every row of $\mathbf{B}$ is a codeword, $\mathbf{v}^T\mathbf{B} = \mathbf{v}^T\mathbf{A}^T G$ is also a codeword. Therefore, one can take any off-the-shelf decoding algorithm for the code whose generator matrix is $\mathbf{G}$, and obtain $\mathbf{v}^T\mathbf{A}^T$. Note that we need fast decoding, as it is performed in every iteration of the gradient computation by the master. As far as we know, any off-the-shelf decoding algorithm (over real numbers) requires at least a quadratic computational complexity, which leads to $\Omega(n^2 + d^2)$ decoding complexity per gradient computation, which could be impractical. The trivial scheme does not exploit the block error pattern which we crucially exploit in our coding scheme to give a $\sim O((n+d)m)$ time decoding per gradient computation, which could be a significant improvement over the trivial scheme, since $m$ typically is much smaller than $n$ and $d$. We also want encoding to be efficient (otherwise it defeats the purpose of data encoding) and our sparse encoding matrix achieves that, but any off-the-shelf error correcting codes (over reals) may not give efficient encoding procedure. Now we explain our coding scheme.

We will think of our encoding matrix as $\mathbf{S} = [\mathbf{S}_1^T\ \mathbf{S}_2^T, \ldots, \mathbf{S}_m^T]$, where each $\mathbf{S}_i$ is a $p \times n_r$ matrix and $pm > n_r$. We will determine the value of $p$ and the entries of $\mathbf{S}$ later. For $i \in [m]$, we store the matrix $\mathbf{S}_i\mathbf{A}$ at the worker node $i$. As described in Section II, computation proceeds as follows: For all $i \in [m]$, master sends $\mathbf{v}$ to worker $i$ and receives $\mathbf{S}_i\mathbf{Av}+\mathbf{e}_i$ back from it. Let $\mathbf{e}_i = [e_{i1}, e_{i2}, \ldots, e_{ip}]^T$. Note that $\mathbf{e}_i = \mathbf{0}$ if the $i$'th node is honest, otherwise can be arbitrary. In order to find the corrupt worker nodes, master equivalently writes $\{\mathbf{S}_i\mathbf{Av} + \mathbf{e}_i\}_{i=1}^m$ as $p$ systems of linear equations.

$$\tilde{h}_i(\mathbf{v}) = \tilde{\mathbf{S}}_i\mathbf{Av} + \tilde{\mathbf{e}}_i, \quad i \in [p] \tag{6}$$

where, for every $i \in [p]$, $\tilde{\mathbf{e}}_i = [e_{1i}, e_{2i}, \ldots, e_{mi}]^T$ with $|\mathsf{supp}(\tilde{\mathbf{e}}_i)| \leq t$, and $\tilde{\mathbf{S}}_i$ is an $m \times n_r$ matrix whose $j$'th row is equal to the $i$'th row of $\mathbf{S}_j$, for every $j \in [m]$.

Observe that $\tilde{\mathbf{S}}_i$'s constitute the encoding matrix $\mathbf{S}$, which we have to design. In the following, we will design these



Authorized licensed use limited to: UCLA Library. Downloaded on August 23,2020 at 06:27:26 UTC from IEEE Xplore. Restrictions apply.

matrices $\tilde{\mathbf{S}}_i$'s (which in turn will determine the encoding matrix $\mathbf{S}$), with the help of another matrix $\mathbf{F}$, which will be used to find the error locations, i.e., identities of the compromised worker nodes. We will design $\mathbf{F}$ (of dimension $k \times m$, where $k < m$) and $\tilde{\mathbf{S}}_i$'s such that

**C.1** $\mathbf{F}\tilde{\mathbf{S}}_i = 0$ for every $i \in [p]$.

**C.2** For any $\mathbf{u} \in \mathbb{R}^m$ such that $|\mathsf{supp}(\mathbf{u})| \leq t$, we can efficiently find all the non-zero locations of $\mathbf{u}$ from $\mathbf{Fu}$.

**C.3** For any $\mathcal{T} \subset [m]$ such that $|\mathcal{T}| \geq (m-t)$, let $\mathbf{S}_\mathcal{T}$ denote the $|\mathcal{T}|p \times n_r$ matrix obtained from $\mathbf{S}$ by restricting it to all the $\mathbf{S}_i$'s for which $i \in \mathcal{T}$. We want $\mathbf{S}_\mathcal{T}$ to be of full column rank.

If we can find such matrices, we can recover the desired MV multiplication $\mathbf{Av}$ exactly as follows. Multiplying (6) by $\mathbf{F}$ and using **C.1** yield $\mathbf{f}_i := \mathbf{F}\tilde{h}_i(\mathbf{v}) = \mathbf{F}\tilde{\mathbf{e}}_i$, for every $i \in [p]$, where $|\mathsf{supp}(\tilde{\mathbf{e}}_i)| \leq t$. A crucial observation: Since the non-zero entries of all the error vectors $\tilde{\mathbf{e}}_i$'s occur in the same set $\mathcal{I} = \bigcup_{i=1}^{p} \mathsf{supp}(\tilde{\mathbf{e}}_i)$, a random linear combination of $\tilde{\mathbf{e}}_i$'s has support equal to $\mathcal{I}$ with probability one, if the coefficients $\alpha_i$'s of the linear combination are chosen from an *absolutely continuous* probability distribution, for example, the Gaussian distribution. Let $\tilde{\mathbf{f}} = \alpha_i \left(\sum_{i=1}^{p} \mathbf{f}_i\right) = \alpha_i \left(\sum_{i=1}^{p} \mathbf{F}\tilde{\mathbf{e}}_i\right) = \mathbf{F}\left(\sum_{i=1}^{p} \alpha_i \tilde{\mathbf{e}}_i\right) = \mathbf{F}\tilde{\mathbf{e}}$, where $\tilde{\mathbf{e}} = \sum_{i=1}^{p} \alpha_i \tilde{\mathbf{e}}_i$. Note that, with probability 1, $\mathsf{supp}(\tilde{\mathbf{e}}) = \mathcal{I}$. **C.2** allows us to find this set efficiently. There are many choices for $\mathbf{F}$ that can handle different levels of sparsity with varying decoding complexity. We can choose any of these matrices depending on our need, and this will not affect the design of our encoding matrix $\mathbf{S}$. Specifically, the choice of $\mathbf{F}$ may only affect the non-zero entries of $\mathbf{S}$, neither their repetitive format, nor their locations; see (7). Note that such an $\mathbf{F}$ gives an error-correcting code over $\mathbb{R}$ and we can use any such code. In particular, we can use a $2t \times m$ Vandermonde matrix along with the Reed-Solomon decoding, which can correct up to $t$ errors and has decoding complexity of $O(m^2)$ [23].

Once we have found the corrupt nodes, we can discard all the information received from them. This yields $\mathbf{S}_\mathcal{T}\mathbf{Av}$, where $\mathcal{T}$ is the set of all honest worker nodes. Since $\mathbf{S}_\mathcal{T}$ is of full column rank (by **C.3**), we can recover $\mathbf{Av}$ from $\mathbf{S}_\mathcal{T}\mathbf{Av}$ exactly. For this, instead of computing $\mathbf{S}_\mathcal{T}^+$, the pseudo inverse of $\mathbf{S}_\mathcal{T}$ (such that $\mathbf{S}_\mathcal{T}^+\mathbf{S}_\mathcal{T}$ is an identity matrix), which is computationally very expensive, we leverage the sparse and repetitive structure of $\mathbf{S}$ to recover $\mathbf{Av}$ efficiently.

**Designing the encoding matrix S.** Let $\mathcal{N}(\mathbf{F}) \subset \mathbb{R}^m$ denote the null-space of $\mathbf{F}$. Since $\mathsf{rank}(\mathbf{F}) = 2t$, dimension of $\mathcal{N}(\mathbf{F})$ is $q = (m - 2t)$. Let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$ be a basis of $\mathcal{N}(\mathbf{F})$, and let $\mathbf{b}_i = [b_{i1}\ b_{i2} \ldots b_{im}]^T$, for every $i \in [q]$. The encoding matrix for the $i$'th worker is the following:

$$\mathbf{S}_i = \begin{bmatrix} b_{1i} \ldots b_{qi} & & & \\ & \ddots & & \\ & & b_{1i} \ldots b_{qi} & \\ & & & b_{1i} \ldots b_{li} \end{bmatrix}_{p \times n_r} \tag{7}$$

Here $q = (m - 2t)$ and $l = n_r - (p-1)q$, where $p = \lceil \frac{n_r}{q} \rceil$. All the unspecified entries of $\mathbf{S}_i$ are zero.


## ACKNOWLEDGEMENTS

The work was partially supported by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196, by the UC-NL grant LFR-18-548554, by the NSF award 1740047, 1527550, 1514531, and by the City University of Hong Kong grant (No. 7200594). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.


## REFERENCES


[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2010, pp. 2595–2603.

[3] T. F. Abdelzaher *et al.*, "Will distributed computing revolutionize peace? the emergence of battlefield IoT," in *ICDCS 2018*, 2018, pp. 1129–1138.

[4] J. Konecný, "Stochastic, distributed and federated optimization for machine learning," Ph.D. dissertation, University of Edinburgh, 2017.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[6] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Adv. in Neural Information Processing Systems*, 2017, pp. 118–128.

[7] L. Chen, H. Wang, Z. B. Charles, and D. S. Papailiopoulos, "DRACO: byzantine-resilient distributed training via redundant gradients," in *International Conference on Machine Learning*, 2018, pp. 902–911.

[8] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *POMACS*, vol. 1, no. 2, pp. 44:1–44:25, 2017.

[9] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[10] D. Data, L. Song, and S. N. Diggavi, "Data encoding for byzantine-resilient distributed optimization," [Available Online].

[11] R. Tibshirani, "Convex optimization notes," http://www.stat.cmu.edu/~ryantibs/convexopt-S15/scribes/08-prox-grad-scribed.pdf, 2015.

[12] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[13] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *In Advances in Neural Information Processing Systems*, 2017, pp. 5440–5448.

[14] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, 2018, pp. 3518–3527.

[15] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[16] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic MDS codes and expander graphs," in *International Conference on Machine Learning*, 2018, pp. 4302–4310.

[17] Z. B. Charles and D. S. Papailiopoulos, "Gradient coding using the stochastic block model," in *IEEE International Symposium on Information Theory*, 2018, pp. 1998–2002.

[18] W. Halbawi, N. A. Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *IEEE International Symposium on Information Theory*, 2018, pp. 2027–2031.

[19] K. Lee, M. Lam, R. Pedarsani, D. S. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[20] S. Dutta, V. R. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Adv. in Neural Information Processing Systems*, 2016, pp. 2092–2100.

[21] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, 2018, pp. 5636–5645.

[22] D. Data, L. Song, and S. N. Diggavi, "Data encoding methods for byzantine-resilient distributed gradient descent," Invited presentation at Allerton 2018.

[23] M. Akçakaya and V. Tarokh, "A frame construction and a universal distortion bound for sparse representations," *IEEE Trans. Signal Processing*, vol. 56, no. 6, pp. 2443–2450, 2008.




Authorized licensed use limited to: UCLA Library. Downloaded on August 23,2020 at 06:27:26 UTC from IEEE Xplore. Restrictions apply.