Gradient Coding from Cyclic MDS Codes and Expander Graphs

Netanel Raviv [1] **Itzhak Tamo** [2] **Rashish Tandon** [3] **Alexandros G. Dimakis** [4]

Abstract

Gradient coding is a technique for straggler mitigation in distributed learning. In this paper we design novel gradient codes using tools from classical coding theory, namely, cyclic MDS codes, which compare favourably with existing solutions, both in the applicable range of parameters and in the complexity of the involved algorithms. Second, we introduce an approximate variant of the gradient coding problem, in which we settle for approximate gradient computation instead of the exact one. This approach enables graceful degradation, i.e., the ℓ_2 error of the approximate gradient is a decreasing function of the number of stragglers. Our main result is that the normalized adjacency matrix of an expander graph can yield excellent approximate gradient codes, and that this approach allows us to perform significantly less computation compared to exact gradient coding. We experimentally test our approach on Amazon EC2, and show that the generalization error of approximate gradient coding is very close to the full gradient while requiring significantly less computation from the workers.

1. Introduction

Data intensive machine learning tasks have become ubiquitous in many real-world applications, and with the increasing size of training data, distributed methods have gained increasing popularity. However, the performance of distributed methods (in synchronous settings) is strongly dictated by *stragglers*, i.e., nodes that are slow to respond or unavailable. In this paper, we focus on coding theoretic (and graph theoretic) techniques for mitigating stragglers in distributed synchronous gradient descent.

The coding theoretic framework for straggler mitigation called *gradient coding* was first introduced in (Tandon et al., 2017). It consists of a system with one master and n worker nodes, in which the data is partitioned into k parts, and one or more parts is assigned to each one of the workers. In turn, each worker computes the partial gradient on each of its assigned partitions, linearly combines the results according to some predetermined vector of coefficients, and sends this linear combination back to the master node. Choosing the coefficients at each node judiciously, one can guarantee that the master node is capable of reconstructing the full gradient even if *any* s machines fail to perform their work. The *storage overhead* of the system, which is denoted by d, refers to the amount of redundant computations, or alternatively, to the number of data parts that are sent to each node (see example in Fig. 1).

The importance of straggler mitigation was demonstrated in a series of recent studies (e.g., (Li et al., 2014) and (Yadwadkar et al., 2016)). In particular, it was demonstrated in (Tandon et al., 2017) that stragglers may run up to $\times 5$ slower than the typical worker performance ($\times 8$ in (Yadwadkar et al., 2016)) on Amazon EC2, especially for the cheaper virtual machines; such erratic behavior is unpredictable and can significantly delay training. One can, of course, use more expensive instances but the goal here is to use coding theoretic methods to provide reliability out of cheap unreliable workers, overall reducing the cost of training.

The work of (Tandon et al., 2017) established the fundamental bound $d \geq s + 1$, provided a deterministic construction which achieves it with equality when $s + 1 | n$, and a randomized one which applies to all s and n. Subsequently, deterministic constructions were also obtained by (Dutta et al., 2016) and (Halbawi, 2017). These works have focused on the scenario where s is known prior to the construction of the system. Furthermore, the exact computation of the full gradient is guaranteed if the number of stragglers is at most s, but no error bound is guaranteed if this number exceeds s.

The contribution of this work is twofold. For the computation of the exact gradient we employ tools from classic coding theory, namely, cyclic MDS codes, in order to ob-

[1]Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, USA. [2]Department of Electrical Engineering–Systems, Tel-Aviv University, Israel. [3]Apple, Seattle, WA, USA. [4]Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA.. Correspondence to: Netanel Raviv <netanel.raviv@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

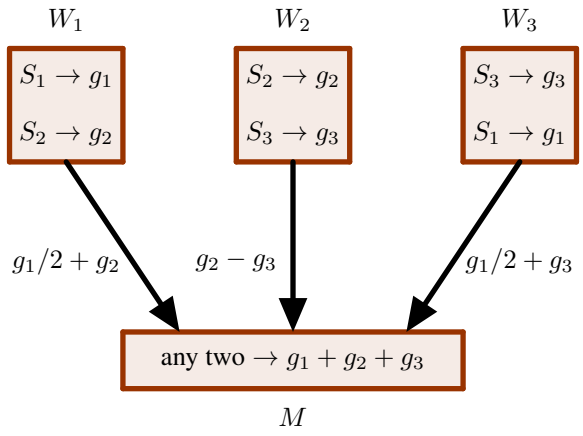


Figure 1. Gradient coding for $n = 3$, $k = 3$, $d = 2$, and $s = 1$ (Tandon et al., 2017). Each worker node W_i obtains two parts S_{i_1}, S_{i_2} of the data set $S = S_1 \cup S_2 \cup S_3$, computes the partial gradients g_{i_1}, g_{i_2}, and sends their linear combination back to the master node M. By choosing the coefficients judiciously, the master node M can compute the full gradient from any two responses, providing robustness against any one straggler.

tain a deterministic construction which compares favourably with existing solutions; both in the applicable range of parameters, and in the complexity of the involved algorithms. Some of these gains are a direct application of well known properties of these codes.

Second, we introduce an *approximate* variant of the gradient coding problem. In this variant, the requirement for exact computation of the full gradient is traded by an approximate one, where the ℓ_2 deviation of the given solution is a decreasing function of the number of stragglers. Note that by this approach, the parameter s is *not* a part of the system construction, and the system can provide an approximate solution for any $s < n$, whose quality deteriorates gracefully as s increases. In the suggested solution, the coefficients at the worker nodes are based on an important family of graphs called *expanders*. In particular, it is shown that setting these coefficients according to a normalized adjacency matrix of an expander graph, a strong bound on the error term of the resulting solution is obtained. Moreover, this approach enables to break the aforementioned barrier $d \geq s + 1$, which is a substantial obstacle in gradient coding, and allows the master node to decode using a very simple algorithm.

This paper is organized as follows. Related work regarding gradient coding (and coded computation in general) is listed in Section 2. A framework which encapsulates all the results in this paper is given in Section 3. Necessary mathematical notions from coding theory and graph theory are given in Section 4. The former is used to obtain an algorithm for exact gradient computation in Section 5, and the latter is used for the approximate one in Section 6. Experimental results are given in Section 7. Many proofs, extensions, and examples are omitted due to space constraints, and are given in the online version of this paper (Raviv et al., 2017).

2. Related Work

The work of Lee et al. (Lee et al., 2017) initiated the use of coding theoretic methods for mitigating stragglers in large-scale learning. This work is focused on linear regression and therefore can exploit more structure compared to the general gradient coding problem that we study here. The work by Li et al. (Li et al., 2016), investigates a generalized view of the coding ideas in (Lee et al., 2017), showing that their solution is a single operating point in a general scheme of trading off latency of computation to the load of communication.

Further closely related work has shown how coding can be used for distributed MapReduce, as well as a similar communication and computation tradeoff (Li et al., 2015; 2018). We also mention the work of (Karakus et al., 2017) which addresses straggler mitigation in linear regression by using a different approach, that is not mutually exclusive with gradient coding. In their work, the data is coded rather than replicated at the master node, and the nodes perform their computation on coded data.

The work by (Dutta et al., 2016) generalizes previous work for linear models (Lee et al., 2017) but can also be applied to general models to yield explicit gradient coding constructions. Our results regarding the exact gradient are closely related to the work by (Halbawi, 2017; Halbawi et al., 2017) which was obtained independently from our work. In (Halbawi, 2017), similar coding theoretic tools were employed in a fundamentally different fashion. Both (Halbawi, 2017) and (Dutta et al., 2016) are comparable in parameters to the randomized construction of (Tandon et al., 2017) and are outperformed by us in a wide range of parameters. A detailed comparison of the theoretical asymptotic behaviour is given in the online version of this paper (Raviv et al., 2017).

None of the aforementioned works studies approximate gradient computations. However, we note that subsequent to this work, two unpublished manuscripts (Charles et al., 2017; Li et al., 2017) study a similar approximation setting and obtain related results albeit using randomized as opposed to deterministic approaches.

3. Framework

This section provides a unified framework which accommodates straggler mitigation in both the exact and approximate gradient computation which follow. In order to distribute the execution of gradient descent from a master node M to n worker nodes $\{W_j\}_{j=1}^{n}$, the training set S is partitioned

Algorithm 1 Gradient Coding

Input: Data $S = \{z_i = (x_i, y_i)\}_{i=1}^m$, number of iterations $T > 0$, learning rate schedule $\eta_t > 0$, straggler tolerances $(s_t)_{t\in[T]}$, a matrix $B \in \mathbb{C}^{n\times n}$, and a function $A : \mathcal{P}(n) \to \mathbb{C}^n$.
Initialize $\boldsymbol{w}^{(1)} = (0, \ldots, 0)$.
Partition $S = \cup_{i=1}^n S_i$ and send $\{S_j : j \in \mathrm{supp}(B_i)\}$ to W_i for every $i \in [n]$.
for $t = 1$ **to** T **do**
  M broadcasts $\boldsymbol{w}^{(t)}$ to all nodes.
  Each W_j sends $\frac{1}{n}\sum_{i\in\mathrm{supp}(B_j)} B_{j,i} \cdot \nabla L_{S_i}(\boldsymbol{w}^{(t)})$ to M.
  M computes $\boldsymbol{v}_t = A(K_t) \cdot \boldsymbol{a}$, where $\boldsymbol{a}_i$ is the response from W_i if it responded and 0 otherwise, for each i.
  M updates $\boldsymbol{w}^{(t+1)} \triangleq \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{v}_t$
end for
return $\boldsymbol{w}^{(T+1)}$.

by M to n disjoint subsets $\{S_i\}_{i=1}^n$ of size[1] $\frac{m}{n}$ each, that are distributed among $\{W_j\}_{i=1}^n$. Each node computes the gradients $\nabla L_{S_i}(\boldsymbol{w})$ of the empirical risks of the S_i-s which it obtained, evaluates them in the current model $\boldsymbol{w}^{(t)}$, and sends some linear combination of the results to M. After obtaining the results of the computation from at least s_t workers, where $(s_t)_{t\in[T]}$ are straggler tolerance parameters, M aggregates them to form the gradient $\nabla L_S(\boldsymbol{w}^{(t)})$ of the overall empirical risk at $\boldsymbol{w}^{(t)}$.

To support mitigation of stragglers in this setting, the following notions are introduced. Let $B \in \mathbb{C}^{n\times n}$ be a matrix whose i-th row B_i contains the coefficients of the linear combination $\sum_{j=1}^n B_{i,j} \cdot \nabla L_{S_j}(\boldsymbol{w}^{(t)})$ that is sent to M by W_i. Note that the support $\mathrm{supp}(B_i)$ contains the indices of the sets S_j that are to be sent to W_i by M. Given a set of non-stragglers $K \in \mathcal{P}(n)$, where $\mathcal{P}(n)$ is the set of all nonempty subsets of $[n]$, a function $A : \mathcal{P}(n) \to \mathbb{C}^n$ provides M with a vector by which the results from $\{W_i\}_{i\in K}$ are to be linearly combined to obtain the vector $\boldsymbol{v}_t$. For convenience of notation, assume that $\mathrm{supp}(A(K)) \subseteq K$ for all $K \in \mathcal{P}(n)$. In most of the subsequent constructions, the matrix B and the function A will be defined over $\mathbb{R}$ rather than over $\mathbb{C}$.

The construction of the matrix B and the function A in Algorithm 1 enables to compute the gradient both *exactly* (which requires the storage overhead d to be at least $s_t + 1$ for all $t \in [T]$) and *approximately*. In what follows, the respective requirements and guarantees from A and B are discussed. In the following definition, for an integer a let $\mathbb{1}_a$ be the vector of a ones, where the subscript is omitted if clear from context.

Definition 1. *A matrix $B \in \mathbb{C}^{n\times n}$ and a function $A : \mathcal{P}(n) \to \mathbb{C}^n$ satisfy the Exact Computation (EC) condition if for all $K \subseteq [n]$ such that $|K| \geq \max_{t\in[T]} s_t$, we have $A(K) \cdot B = \mathbb{1}$. Further, for a non-decreasing function $\epsilon : [n-1] \to \mathbb{R}_{\geq 0}$ such that $\epsilon(0) = 0$, A and B satisfy the ϵ-Approximate Computation (ϵ-AC) condition, if for all $K \in \mathcal{P}(n)$, we have $d_2(A(K)B, \mathbb{1}) \leq \epsilon(|K^c|)$ (where d_2 is the ordinary Euclidean distance).*

Notice that the error term ϵ in the above definition is a function since it is required to decrease with the number of stragglers. The conditions which are given in Definition 1 guarantee the exact and approximate computation by the following lemmas, whose proofs are given in (Raviv et al., 2017).

Lemma 2. *If A and B satisfy the EC condition, then for all $t \in [T]$ we have $\boldsymbol{v}_t = \nabla L_S(\boldsymbol{w}^{(t)})$.*

The next lemma bounds the deviance of $\boldsymbol{v}_t$ from the gradient of the empirical risk at the current model $\boldsymbol{w}^{(t)}$ by using the function ϵ and the *spectral norm* $\|\cdot\|_{spec}$ of the matrix of empirical losses.

Lemma 3. *For a function ϵ as above, if A and B satisfy the ϵ-AC condition, then $d_2(\boldsymbol{v}_t, \nabla L_S(\boldsymbol{w}^{(t)})) \leq \epsilon(|K_t^c|) \cdot \|N(\boldsymbol{w}^{(t)})\|_{spec}$.*

Due to Lemma 2 and Lemma 3, in the remainder of this paper we focus on constructing A and B that satisfy either the EC condition (Section 5) or the ϵ-AC condition (Section 6).

4. Mathematical Notions

This section provides a brief overview on the mathematical notions that are essential for the suggested schemes. The exact computation (Sec. 5) requires notions from coding theory, and the approximate one (Sec. 6) requires notions from graph theory. The coding theoretic material in this section is taken from (Roth, 2006), which focuses on finite fields, and yet the given results extend verbatim to the real or complex case (see also (Marshall, 1984), Sec. 8.4).

For $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}\}$ an $[n, \kappa]$ (linear) code C over $\mathbb{F}$ is a subspace of $\mathbb{F}^n$. The minimum distance δ of C is $\min\{d_H(x, y) : x, y \in C,\ x \neq y\}$, where d_H denotes the *Hamming distance* $d_H(x_i, y_i) = |\{i|x_i \neq y_i\}|$. Note that the minimum distance of a code is equal to its minimum *Hamming weight* $w_H(x) = \|x\|_0 = |\mathrm{supp}(x)|$. The well-known *Singleton* bound states that $\delta \leq n - \kappa + 1$, and codes which attain this bound with equality are called Maximum Distance Separable (MDS) codes. A code C is called *cyclic* if the cyclic rotation of any codeword in C is yet another

[1] For simplicity, assume that $m|n$. The given scheme could be easily adapted to the case $m \nmid n$. Further, the assumption that the number of partitions equals to the number of nodes is a mere convenience, and all subsequent schemes may be easily adapted to the case where the number of partitions is at most the number of nodes.

codeword in C. The *dual* of C is $C^\perp \triangleq \{y \in \mathbb{F}^n | y \cdot c^\top = 0 \text{ for all } c \in C\}$. Several well-known and easy to prove properties of MDS codes are used throughout this paper.

Lemma 4. *If $C \subseteq \mathbb{F}^n$ is an $[n, \kappa]$ MDS code, then*

A1. $C^\perp$ is an $[n, n-\kappa]$ MDS code, and hence its minimum Hamming weight is $\kappa + 1$.

A2. For any subset $K \subseteq [n] \triangleq \{1, \ldots, n\}$ of size $n - \kappa + 1$ there exists a codeword in C whose support (i.e., the set of nonzero indices) is K.

A3. The reverse code $C^R \triangleq \{(c_n, \ldots, c_1) | (c_1, \ldots, c_n) \in C\}$ is an $[n, \kappa]$ MDS code.

Two common families of codes are used in the sequel—Reed-Solomon (RS) codes and Bose-Chaudhuri-Hocquenghem (BCH) codes. An RS code C of length n, dimension s, and pairwise distinct *evaluation points* $\{\alpha_i\}_{i=0}^{n-1} \subseteq \mathbb{F}$ is defined as $C = \{(f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{n-1})) : f \in \mathbb{F}^{<s}[x]\}$, where $\mathbb{F}^{<s}[x]$ is the set of polynomials of degree less than s and coefficients from $\mathbb{F}$. Alternatively, RS codes can be defined as the left image of a Vandermonde matrix on $\{\alpha_i\}_{i=0}^{n-1}$. It is widely known that RS codes are MDS codes, and in some cases, they are also cyclic.

In contrast with RS codes, a codeword of a BCH code is considered as a polynomial. That is, a codeword $c = (c_0, c_1, \ldots, c_{n-1})$ is identified by the univariate polynomial $c(x) \triangleq c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$. For a set of complex numbers $A \subseteq \mathbb{C}$, the (real) BCH code on A is the set of polynomials The set A on which a given BCH code C vanishes is called *the root set of* C. For some sets A, the resulting codes are cyclic.

Lemma 5. *(Marshall, 1984; Roth, 2006) If the root set A of a BCH code C of length n consists of n-th roots of unity, then C is cyclic.*

Proof. If $c(x)$ is a codeword in C, then its cyclic shift is given by $\tilde{c}(x) \triangleq c(x) \cdot x \bmod (x^n - 1) = x \cdot c(x) - c_{n-1} \cdot (x^n - 1)$. Since the root set consists of n-th roots of unity, it follows that for any $\alpha \in A$,

$$\tilde{c}(\alpha) = \alpha \cdot c(\alpha) - c_{n-1} \cdot (\alpha^n - 1) = \alpha \cdot c(\alpha) = 0,$$

and hence $\tilde{c}$ is a codeword in C. □

Further, the structure of A may also imply a lower bound on the distance of C.

Theorem 6. *(The BCH bound) (Marshall, 1984; Roth, 2006) If A contains a subset of D consecutive powers of a primitive root of unity (i.e., a subset of the form $\omega^b, \omega^{b+1}, \ldots, \omega^{b+D-1}$, where ω is an n-th root of unity of multiplicative order n), then the minimum distance of C is at least $D + 1$.*

In the remainder of this section, a brief overview about expander graphs is given. The interested reader is referred to (Hoory et al., 2006) for further details. Let $G = (V, E)$ be a d-regular, undirected, and connected graph on n nodes. Let $A_G \in \mathbb{R}^{n \times n}$ be the *adjacency matrix* of G, i.e., $(A_G)_{i,j} = 1$ if and only if $\{i, j\} \in E$. Since A_G is a real symmetric matrix, it follows that it has real eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, and denote $\lambda \triangleq \max\{|\lambda_2|, |\lambda_n|\}$. It is widely known (Hoory et al., 2006) that $\lambda_1 = d$, and that $\lambda_n \geq -d$, where equality holds if and only if G is bipartite. Further, it also follows from A_G being real and symmetric that it has a basis of orthogonal real eigenvectors $v_1 = \mathbb{1}, v_2, \ldots, v_n$, and w.l.o.g assume that $\|v_i\|_2 = 1$ for every $i \geq 2$. The parameters λ and d are related by the famous Alon-Boppana Theorem.

Theorem 7. *(Hoory et al., 2006) Any d regular graph on n vertices satisfies that $\lambda \geq 2\sqrt{d-1} - o_n(1)$, where $o_n(1)$ is an expression which tends to zero as n tends to infinity.*

Constant degree regular graphs (i.e., families of graphs with fixed degree d that does not depend on n) for which λ is small in comparison with d are largely referred to as *expanders*. In particular, graphs which attain the above bound asymptotically (i.e., $\lambda \leq 2\sqrt{d-1}$) are called *Ramanujan graphs*, and several efficient constructions are known (Lubotzky et al., 1988; Cohen, 2016).

5. Exact Gradient Computation from Cyclic MDS Codes

For a given n and s, let C be a cyclic $[n, n-s]$ MDS code over $\mathbb{F}$ that contains $\mathbb{1}$ (explicit codes are given in the sequel). According to Lemma 4, there exists a codeword $c_1 \in C$ whose support is $\{1, \ldots, s+1\}$. Let $c_2, \ldots, c_n$ be all cyclic shifts of c_1, which lie in C by its cyclic property. Finally, let B be the $n \times n$ matrix whose *columns* are $c_1, \ldots, c_n$, i.e., $B \triangleq (c_1^\top, c_2^\top, \ldots, c_n^\top)$.

Lemma 8. *The matrix B satisfies the following properties.*

B1. $\|b\|_0 = s + 1$ for every row b of B.

B2. Every row of B is a codeword in C^R.

B3. The column span of B is the code C.

B4. Every set of $n - s$ rows of B are linearly independent over $\mathbb{F}$.

Proof. To prove B1 and B2, observe that B is of the follow-

ing form, where $c_1 \triangleq (\beta_1, \ldots, \beta_{s+1}, 0, \ldots, 0)$.

$$\begin{pmatrix} \beta_1 & 0 & \cdots & 0 & \beta_{s+1} & \beta_s & \ldots & \beta_2 \\ \beta_2 & \beta_1 & 0 & \cdots & 0 & \beta_{s+1} & \ldots & \beta_3 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ \beta_s & \beta_{s-1} & \cdots & \beta_1 & 0 & \cdots & 0 & \beta_{s+1} \\ \beta_{s+1} & \beta_s & \cdots & \beta_2 & \beta_1 & 0 & \cdots & 0 \\ 0 & \beta_{s+1} & \cdots & \beta_3 & \beta_2 & \beta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \beta_{s+1} & \beta_s & \beta_{s-1} & \cdots & \beta_1 \end{pmatrix}.$$

To prove B3, notice that the leftmost $n-s$ columns of B have leading coefficients in different positions, and hence they are linearly independent. Thus, the dimension of the column span of B is at least $n-s$, and since $\dim C = n-s$, the claim follows.

To prove B4, assume for contradiction that there exist a set of $n-s$ linearly dependent rows. Hence, there exists a vector $v \in \mathbb{F}^n$ of Hamming weight $n-s$ such that $vB = 0$. According to B3, the columns of B span C, and hence the vector v lies in the dual $C^\perp$ of C. Since $C^\perp$ is an $[n, s]$ MDS code by Lemma 4, it follows that the minimum Hamming weight of a codeword in $C^\perp$ is $n-s+1$, a contradiction. □

Since C^R is of dimension $n-s$, it follows from parts B2 and B4 of Lemma 8 that every set of $n-s$ rows of B are a basis to C^R. Furthermore, since $\mathbb{1} \in C$ it follows that $\mathbb{1} \in C^R$. Therefore, there exists a function $A : \mathcal{P}(n) \to \mathbb{F}^n$ such that for any set $K \subseteq [n]$ of size $n-s$ we have that $\mathrm{supp}(A(K)) = K$ and $A_K \cdot B = \mathbb{1}$.

Theorem 9. . *The above A and B satisfy the EC condition (Definition 1).*

In the remainder of this section, two cyclic MDS codes over the complex numbers and the real numbers are suggested, from which the construction in Theorem 9 can be obtained. These constructions are taken from (Marshall, 1984) (Sec. II.B), and are given with a few adjustments to our case. The contributions of these codes is summarized in the following theorem, and omitted proofs are given in (Raviv et al., 2017).

Theorem 10. *For any given n and s there exist explicit complex matrices A and B that satisfy the EC-condition with optimal $d = s+1$. The respective encoding (i.e., constructing B) and decoding (i.e., constructing $A(K)$ given K) complexities are $O(s(n-s))$ and $O(s\log^2 s + n\log n)$, respectively. Over $\mathbb{R}$, for any given n and s such that $n \neq s \bmod 2$ there exist explicit matrices A and B that satisfy the EC-condition with optimal $d = s+1$. The encoding and decoding complexities are $O(\min\{s\log^2 s, n\log n\})$ and $O(g_s + s(n-s))$, where g_s is the complexity of inverting a generalized Vandermonde matrix.*

5.1. Cyclic-MDS Codes Over the Complex Numbers

For a given n and s, let $i = \sqrt{-1}$, and let $A \triangleq \{\alpha_j\}_{j=0}^{n-1}$ be the set of complex roots of unity of order n, i.e., $\alpha_j \triangleq e^{2\pi ij/n}$. Let $G \in \mathbb{C}^{(n-s)\times n}$ be a complex Vandermonde matrix over A, i.e., $G_{k,j} = \alpha_j^k$ for any $j \in \{0, 1, \ldots, n-1\}$ and any $k \in \{0, 1, \ldots, n-s-1\}$. Finally, let $C \triangleq \{xG | x \in \mathbb{C}^{n-s}\}$. It is readily verified that C is an $[n, n-s]$ MDS code that contains $\mathbb{1}$, whose codewords may be seen as the evaluations of all polynomials in $\mathbb{C}^{<n-s}[x]$ on the set A.

Lemma 11. *C is a cyclic code.*

Corollary 12. *The code C is a cyclic MDS code which contains $\mathbb{1}$, and hence it can be used to obtain the matrices A and B, as described in Theorem 9.*

Given a set K of $n-s$ non-stragglers, an algorithm for computing the encoding vector $A(K)$ in $O(s\log^2 s + n\log n)$ operations over $\mathbb{C}$ (after a one-time initial computation of $O(s^2 + s(n-s))$), is given in Appendix B. The complexity of this algorithm is asymptotically smaller than the corresponding algorithm in (Dutta et al., 2016) and (Halbawi, 2017) whenever $s = o(n)$. Furthermore, the cyclic structure of the matrix B enables a very simple algorithm for its construction; this algorithm compares favorably with previous works for any s, and is given in Appendix B as well.

5.2. Cyclic-MDS Codes Over the Real Numbers

If one wishes to abstain from using complex numbers, e.g., in order to reduce bandwidth, we suggest the following construction, which provides a cyclic MDS code over the reals. This construction relies on (Marshall, 1984) (Property 3), with an additional specialized property.

Construction 13. *For a given n and s such that $n \neq s \bmod 2$, define the following BCH codes over the reals. In both cases denote $\omega \triangleq e^{2\pi i/n}$.*

1. *If n is even and s is odd let $s' \triangleq \lfloor \frac{s}{2} \rfloor$, and let C_1 be a BCH code which consists of all polynomials in $\mathbb{R}^{<n}[x]$ that vanish over the set $A_1 \triangleq \{\omega^{n/2-s'}, \omega^{n/2-s'+1}, \ldots, \omega^{n/2+s'}\}$.*
2. *If n is odd and s is even let $n' \triangleq \lfloor \frac{n}{2} \rfloor$, and let C_2 be a BCH code which consists of all polynomials in $\mathbb{R}^{<n}[x]$ that vanish over the set $A_2 \triangleq \{\omega^{n'-s/2+1}, \omega^{n'-s/2+2}, \ldots, \omega^{n'+s/2}\}$.*

Lemma 14. *The codes C_1 and C_2 from Construction 13 are cyclic $[n, n-s]$ MDS codes that contain $\mathbb{1}$.*

Algorithms for computing the matrix B and the vector $A(K)$ for the codes in this subsection are given in Appendix C. The algorithm for construction B outperforms previous works whenever $s = o(n)$, and the algorithm for computing $A(K)$

outperforms previous works for a smaller yet wide range of s values.

6. Approximate Gradient Computation from Expander Graphs

Setting B as the identity matrix and A as the function which maps every $K \in \mathcal{P}(n)$ to its binary characteristic vector $\mathbb{1}_K$, clearly satisfies the ϵ-AC scheme for $\epsilon(K) = \sqrt{|K^c|}$, since

$$d_2(A(K)B, \mathbb{1}) = d_2(\mathbb{1}_K, \mathbb{1}) = \sqrt{|K^c|}. \tag{1}$$

It is readily verified that this approach (termed hereafter as "trivial") amount to *ignoring* the stragglers, which is essentially equivalent to (Chen et al., 2016). We show that this can be outperformed by setting B to be a normalized adjacency matrix of a connected regular graph on n nodes, which is constructed by the master before dispersing the data, and setting A to be some carefully chosen yet simple function.

The resulting error function $\epsilon(s)$ depends on the parameters of the graph, whereas the resulting storage overhead d is given by its *degree* (i.e., the *fixed* number of neighbors of each node). The error function is given below for a general connected and regular graph, and particular examples with their resulting errors are given in the sequel. In particular, it is shown that taking the graph to be an expander graph provides a deviation ϵ which is asymptotically less than $\sqrt{s}$ (Eq. (1)) whenever $s = o(n)$. In some cases, smaller deviation is also obtained for larger values of s.

For a given n let G be a connected and d-regular graph on n nodes, with eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$ and corresponding eigenvectors $v_1 = \mathbb{1}, v_2, \ldots, v_n$ as described in Subsection 4. Let $B \triangleq \frac{1}{d} \cdot A_G$, and for a given $K \subseteq [n]$ of size $n - s$, define $u_K \in \mathbb{R}^n$ as

$$(u_K)_i = \begin{cases} -1 & i \notin K \\ \frac{s}{n-s} & i \in K \end{cases}. \tag{2}$$

Lemma 15. *For any $K \subseteq [n]$ of size $n - s$, $u_K \in \langle v_2, \ldots, v_n \rangle$.*

Proof. First, observe that $\langle v_2, \ldots, v_n \rangle$ is exactly the subspace of all vectors whose sum of entries is zero. This follows from the fact that $\{\mathbb{1}, v_2, \ldots, v_n\}$ is an orthogonal basis, hence $v_i \cdot \mathbb{1} = 0$ for every $i \geq 2$, and from the fact that $\{v_2, \ldots, v_n\}$ are linearly independent. Since the sum of entries of u_K is zero, the result follows. □

Corollary 16. *For any $K \subseteq [n]$ there exists $\alpha_2, \ldots, \alpha_n \in \mathbb{R}$ such that $u_K = \alpha_2 v_2 + \ldots + \alpha_n v_n$, and $\|u_K\|_2 = \sqrt{\sum_{i=2}^n \alpha_i^2} = \sqrt{\frac{ns}{n-s}}$.*

Proof. The first part follows immediately from Lemma 15. The second part follows by computing the ℓ_2 norm of u_K in two ways, once by its definition (2) and again by using the representation of u_K as a linear combination of the orthonormal set $\{v_2, \ldots, v_n\}$. □

Now, define $A : \mathcal{P}(n) \to \mathbb{R}^n$ as $A(K) = u_K + \mathbb{1}$, and observe that $\mathrm{supp}(A(K)) = K$ for all $K \in \mathcal{P}(n)$. Note that computing $A(K)$ given K is done by a straightforward $O(n)$ algorithm. The error function ϵ is given by the following lemma.

Lemma 17. *For every set $K \subseteq [n]$ of size $n - s$, $d_2(A_K B, \mathbb{1}) \leq \frac{\lambda}{d} \cdot \sqrt{\frac{ns}{n-s}} \triangleq \epsilon(s)$.*

Proof. Notice that the eigenvalues of B are $\mu_i \triangleq \frac{\lambda_i}{d}$, and hence $\mu \triangleq \max\{|\mu_2|, |\mu_n|\}$ equals $\frac{\lambda}{d}$. Further, the eigenvectors are identical to those of A_G. Therefore, it follows from Corollary 16 that

$$\begin{aligned} d_2(A_K B, \mathbb{1}) &= d_2((\mathbb{1} + u_K)B, \mathbb{1}) \\ &= d_2((\mathbb{1} + \alpha_2 v_2 + \ldots + \alpha_n v_n)B, \mathbb{1}) \\ &= d_2(\mathbb{1} + \alpha_2 \mu_2 v_2 + \ldots + \alpha_n \mu_n v_n, \mathbb{1}) \\ &= \|\alpha_2 \mu_2 v_2 + \ldots + \alpha_n \mu_n v_n\|_2, \end{aligned}$$

and since $\{v_2, \ldots, v_n\}$ are orthonormal, it follows that

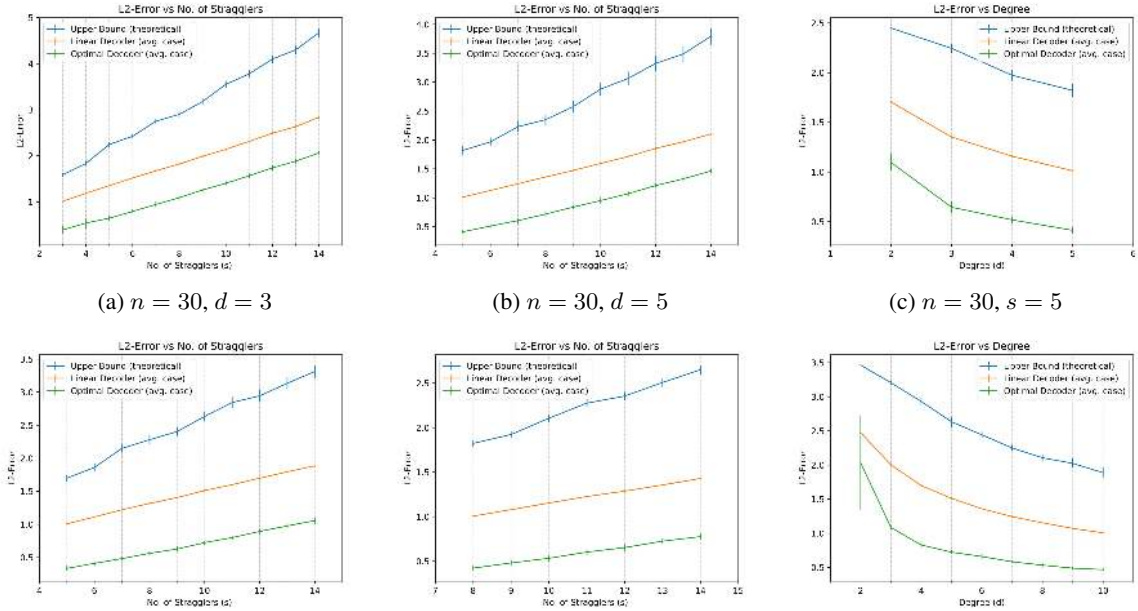
$$\begin{aligned} &\|\alpha_2 \mu_2 v_2 + \ldots + \alpha_n \mu_n v_n\|_2 \\ &\quad = \sqrt{\sum_{i=2}^n \mu_i^2 \alpha_i^2} \leq \sqrt{\sum_{i=2}^n \mu^2 \alpha_i^2} \\ &\quad = \mu \sqrt{\sum_{i=2}^n \alpha_i^2} = \frac{\lambda}{d} \sqrt{\frac{ns}{n-s}}. \end{aligned}$$

□

Corollary 18. *The above A and B satisfy the ϵ-AC condition for $\epsilon(s) = \frac{\lambda}{d}\sqrt{\frac{ns}{n-s}}$. The storage overhead of this scheme equals the degree d of the underlying regular graph G.*

It is evident that in order to obtain small deviation $\epsilon(s)$, it is essential to have a small λ and a large d. However, most constructions of expanders have focused in the case were d is constant (i.e., $d = O(1)$). On one hand, constant d serves our purpose well since it implies a constant storage overhead. On the other hand, a constant d does not allow λ/d to tend to zero as n tends to infinity due to Theorem 7.

To present the contribution of the suggested scheme, it is compared to the trivial one. Clearly, for any given number of stragglers s, it follows from (1) and from Lemma 17 that the latter scheme outperforms the trivial one if



(a) $n = 30$, $d = 3$ (b) $n = 30$, $d = 5$ (c) $n = 30$, $s = 5$

(d) $n = 50$, $d = 5$ (e) $n = 50$, $d = 8$ (f) $n = 50$, $s = 5$

Figure 2. ℓ_2-error for recovery of $\mathbb{1}$ using normalized adjacency matrices of random d-regular graphs.

$$\frac{\lambda}{d}\sqrt{\frac{n}{n-s}} < 1. \tag{3}$$

Since any connected and non-bipartite graph satisfies that $\lambda < d$, it follows that Eq. (3) holds asymptotically for any $s = o(n)$. The following example shows the improved error rate for Margulis graphs (given in (Hoory et al., 2006), Sec. 8), that are rather easy to construct. Several additional examples for Ramanujan graphs, which attain much better error rate but are harder to construct, are given in Appendix A.

Example 19. *For any integer n there exists an 8-regular graph on n nodes with $\lambda \leq 5\sqrt{2}$. For example, by using these graphs with the parameters $n = 500$, $d = 8$, $s = 50$, we have that $\epsilon(s) = \frac{\lambda}{d}\sqrt{\frac{ns}{n-s}} \leq \frac{5\sqrt{2}}{8}\sqrt{\frac{500 \cdot 50}{500-50}} \approx 6.59$, whereas $\sqrt{s} \approx 7.07$, an improvement of approximately 6.8%.*

Restricting d to be a constant (i.e., not to grow with n) is detrimental to the error term in (3) due to Theorem 7, but allows lower storage overhead. If one wishes a lower error term at the price of higher overhead, the following is useful.

Example 20. *(Bilu & Linial, 2006) There exists a polynomial algorithm (in n) to produce a graph G with the parameters $(n, d, \lambda) = (2^m, m-1, \sqrt{m \log^3 m})$. For this family of graphs, the relative error term (3) goes to zero as n goes to infinity for $s = \delta n$, $0 < \delta < 1$.*

We also note that for bipartite expanders, for whom $\lambda = d$, can be employed in a slightly different fashion to achieve smaller error terms. The analysis relies on the *singular values* of its adjacency matrix, and the details are in Appendix D. Finally, we have the following lower bound on approximation error of any *Approximate Computation (AC)* scheme, that establishes asymptotic optimality (up to constants) of our earlier proposed scheme, when used with Ramanujan graphs. The proof of this bound is deferred to Appendix E.

Lemma 21. *Consider any $B \in \mathbb{R}^{n \times n}$ with each row having at most d non-zeros. Then, for any $s > d$ there exists a set $K \subseteq [n]$ of size $n - s$ such that*

$$\min_{\substack{a \in \mathbb{R}^n \\ \text{supp}(a) \subseteq K}} d_2(aB, \mathbb{1}) \geq \sqrt{\left\lfloor \frac{s}{d} \right\rfloor} \tag{4}$$

7. Experimental Results

In this section, we present results of experiments on our proposed approximate gradient coding schemes.

7.1. ℓ_2 Error

We measured the performance of our approximate coding schemes in terms of the ℓ_2-error for recovery of the all $\mathbb{1}$s vector. We chose the normalized adjacency matrix of a random d-regular graph (on n vertices) as the matrix B in our schemes. We randomly chose $n - s$ rows of B to be

the surviving workers in any particular iteration, where s is the number of stragglers. For the decoding vector A_K, we chose the vector specified in our schemes in Eq. (2) (called the *Linear decoder* here) as well as the optimal least squares solution (called the *Optimal decoder* here), given as:

$$A_K = \min_a \|aB(K,:) - \mathbb{1}\|_2$$

Note that even though we have no additional theoretical guarantees for the optimal decoder, it is always possible to compute it.

Figure 2 presents the results using graphs on $n = 30, 50$ vertices, and various values of s and d. The results shown are averaged over multiple samples of K and multiple draws of the matrix B.

Figures 2a, 2b, 2d and 2e show ℓ_2-error vs no. of stragglers s. As the no. of stragglers increases, the recovery gets worse for a fixed computation budget (or degree) d. Figures 2c and 2f show ℓ_2-error vs degree d. As the computation budget, d, increases, the recovery error gets better, for a fixed no. of stragglers s. Also, as expected, in all cases, the *Optimal decoder* does better than the *Linear decoder* in terms of ℓ_2-error. Interestingly, we can also observe that on average both the *Linear decoder* and the *Optimal decoder* are better than the theoretical upper bound in our paper. One could even think of exploiting this empirically by randomizing the assignment of the rows of B to the different workers in every iteration.

7.2. Generalization Error

In this section, our *Approximate Gradient Coding* (*Approximate Gradient Coding*) scheme is compared to other baseline approaches. We compare against *Gradient Coding* (*Gradient Coding*) (Tandon et al., 2017), as well as the *Ignore Stragglers* (*Ignoring Stragglers*) approach, where the data is divided equally among all workers, but the master only uses the first $n - s$ gradients.

We measured the performance of our coding schemes in terms of AUC on a validation set for a logistic regression problem, on a real dataset. The dataset we used was the Amazon Employee dataset from Kaggle. We used $26,200$ training samples, and a model dimension of $241,915$ (after one-shot encoding with interaction terms), and used gradient descent to train the logistic regression. For *Gradient Coding* we used a constant learning rate, chosen using cross-validation. For *Approximate Gradient Coding* and *Ignoring Stragglers*, we used a learning rate of $c_1/(t + c_2)$, which is typical for SGD, where c_1 and c_2 were also chosen via cross-validation.

All our methods were implemented in python using MPI4py (similar to (Tandon et al., 2017)). We ran our experiments using `t2.micro` worker instance types on Amazon EC2 and a `c3.8xlarge` master instance type. The results for $n = 30, 50$ are given in Fig. 3 and Fig, 4, in which *Approximate Gradient Coding* corresponds to our approximation schemes with the optimal decoder, whereas *Approximate Gradient Coding* (Linear), termed *Approximate Gradient Coding (Linear)* is our full proposed approximation scheme.

We observe that both these approaches are only slightly worse than *Gradient Coding*, which utilizes the full gradient, and are quite better than the *Ignoring Stragglers* approach. Compared to each other, *Approximate Gradient Coding* and *Approximate Gradient Coding (Linear)* seem equivalent, however *Approximate Gradient Coding* was marginally better. That being said, *Approximate Gradient Coding (Linear)* can be faster since computing the Linear decoder only requires $O(n)$ time, in contrast to $O(n^3)$ time for the optimal decoder.

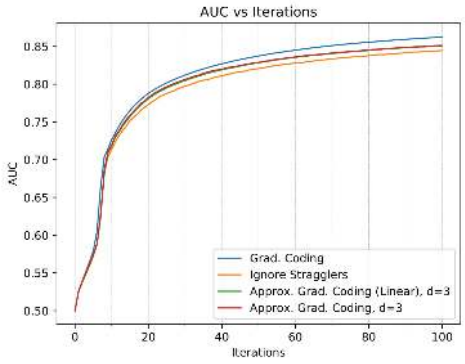


Figure 3. Generalization error vs No. of iterations using $n = 30$ `t2.micro` worker instances on EC2, with $d = 3$, and $s = 5$. Note that in case of *Gradient Coding* (Tandon et al., 2017), the computational overhead here is $\times 6$ times (instead of $\times 3$ in our approach).

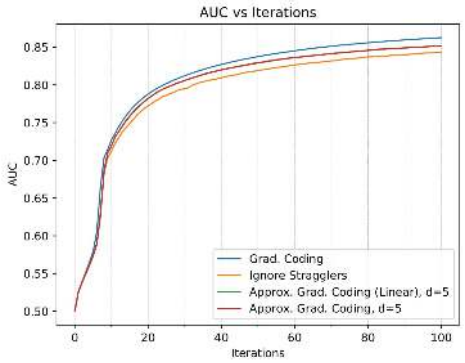


Figure 4. Generalization error vs No. of iterations using $n = 50$ `t2.micro` worker instances on EC2, with $d = 5$, and $s = 10$. Note that in case of *Gradient Coding* (Tandon et al., 2017), the computational overhead here is $\times 11$ times (instead of $\times 5$ in our approach).

Acknowledgments

This research has been supported by NSF Grants CCF 1422549, 1618689, DMS 1723052, ARO YIP W911NF-14-1-0258 and research gifts by Google, Western Digital and NVIDIA. The work of Rashish Tandon was done while he was at UT Austin, prior to joining apple. The work of Itzhak Tamo and Netanel Raviv was supported in part ISF Grant 1030/15 and NSF-BSF Grant 2015814. The work of Netanel Raviv was supported in part by the postdoctoral fellowship of the Center for the Mathematics of Information (CMI), Caltech, and in part by the Lester-Deutsch postdoctoral fellowship.

References

Bilu, Y. and Linial, N. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519, 2006.

Charles, Z., Papailiopoulos, D., and Ellenberg, J. Approximate gradient coding via sparse random graphs. *arXiv preprint arXiv:1711.06771*, 2017.

Chen, J., Monga, R., Bengio, S., and Jozefowicz, R. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

Cohen, M. B. Ramanujan graphs in polynomial time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pp. 276–281. IEEE, 2016.

Dutta, S., Cadambe, V., and Grover, P. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

Halbawi, W. *Error-Correcting Codes for Networks, Storage and Computation*. PhD thesis, California Institute of Technology, 2017.

Halbawi, W., Ruhi, N. A., Salehi, F., and Hassibi, B. Improving distributed gradient descent using reed-solomon codes. *CoRR*, abs/1706.05436, 2017. URL http://arxiv.org/abs/1706.05436.

Hoory, S., Linial, N., and Wigderson, A. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

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

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 2017.

Li, M., Andersen, D. G., Smola, A., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 19–27, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2968826.2968829.

Li, S., Maddah-Ali, M. A., and Avestimehr, A. S. Coded mapreduce. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pp. 964–971. IEEE, 2015.

Li, S., Maddah-Ali, M. A., and Avestimehr, A. S. A unified coding framework for distributed computing with straggling servers. In *Globecom Workshops (GC Wkshps), 2016 IEEE*, pp. 1–6. IEEE, 2016.

Li, S., Kalan, S. M. M., Avestimehr, A. S., and Soltanolkotabi, M. Near-optimal straggler mitigation for distributed gradient methods. *arXiv preprint arXiv:1710.09990*, 2017.

Li, S., Maddah-Ali, M. A., Yu, Q., and Avestimehr, A. S. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2018.

Lubotzky, A., Phillips, R., and Sarnak, P. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

Marshall, T. Coding of real-number sequences for error correction: A digital signal processing problem. *IEEE Journal on Selected Areas in Communications*, 2(2):381–392, 1984.

Raviv, N., Tamo, I., Tandon, R., and Dimakis, A. G. Gradient coding from cyclic MDS codes and expander graphs. *CoRR*, abs/1707.03858, 2017. URL http://arxiv.org/abs/1707.03858.

Roth, R. *Introduction to coding theory*. Cambridge University Press, 2006.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient coding: Avoiding stragglers in distributed learning. In Langley, P. (ed.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3368–3376, Stanford, CA, 2017. Morgan Kaufmann. URL http://proceedings.mlr.press/v70/tandon17a.html.

Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., and Katz, R. Multi-task learning for straggler avoiding predictive job scheduling. *Journal of Machine Learning Research*, 17(106):1–37, 2016. URL http://jmlr.org/papers/v17/15-149.html.