Lightweight Projective Derivative Codes 
for Compressed Asynchronous Gradient Descent

Pedro Soto 1  Ilia Ilmer 1  Haibin Guan 2  Jun Li 3

Abstract

Coded distributed computation has become common practice for performing gradient descent on large datasets to mitigate stragglers and other faults. This paper proposes a novel algorithm that encodes the partial derivatives themselves and furthermore optimizes the codes by performing lossy compression on the derivative codewords by maximizing the information contained in the codewords while minimizing the information between the codewords. The utility of this application of coding theory is a geometrical consequence of the observed fact in optimization research that noise is tolerable, sometimes even helpful, in gradient descent based learning algorithms since it helps avoid overfitting and local minima. This stands in contrast with much current conventional work on distributed coded computation which focuses on recovering all of the data from the workers. A second further contribution is that the low-weight nature of the coding scheme allows for asynchronous gradient updates since the code can be iteratively decoded; i.e., a worker’s task can immediately be updated into the larger gradient. The directional derivative is always a linear function of the direction vectors; thus, our framework is robust since it can apply linear coding techniques to general machine learning frameworks such as deep neural networks.

1. Motivation

The majority of machine learning problems take the form: find a function \( h_{w_0,...,w_k} \) in some family of hypothesis functions \( \mathcal{H} \) that are parameterized over the \( w_0,...,w_k \) which best explains the data.

\[
D = \begin{bmatrix} x_0 & \ldots & x_u & y_0 & \ldots & y_v \\ D_0 & x_1 & \ldots & x_u & y_1 & \ldots & y_v \\ D_1 & x_1 & \ldots & x_u & y_1 & \ldots & y_v \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ D_N & x_1^{(N)} & \ldots & x_u^{(N)} & y_1^{(N)} & \ldots & y_v^{(N)} \end{bmatrix},
\]

where the \( D_i \) are the datapoints, the \( x_i \) are the input features, and the \( y_i \) are the output features.

If the \( h_{w_0,...,w_k} \) are smoothly parameterized by the \( w_0,...,w_k \), then this is usually accomplished by performing gradient descent on the summation of loss functions of the form \( l_i(w) = l(h_w(x^{(i)}), y^{(i)}) \) to compute

\[
\min_{w \in \mathcal{W}} \mathcal{L}(D, w) \overset{\text{def}}{=} \min_{w \in \mathcal{W}} \sum_{D_i} l \left( h_{w}(x^{(i)}), y^{(i)} \right) = \min_{w \in \mathcal{W}} \sum_{D_i} l_i(w);
\]

i.e., find the \( w \) that best fits \( D \). For example, one of the most common loss functions, \( l(f, y) = \frac{1}{2} \| f - y \|^2 \), gives us the mean squared error and the ubiquitous method of least squares.

If the dataset \( D \) has many datapoints \( D_i \), then the overall computation, or job, is distributed as tasks amongst workers, which model a distributed network of computing devices. This solution creates a new problem; stragglers and other faults can severely impact the performance and overall training time. An emerging technique is to use distributed coded computation to mitigate stragglers and other failures in the network. Many of the current algorithms only encode the data; this paper proposes further encoding the directional derivatives as well in such a way that allows for asynchronous gradient updates using low weight codes. Furthermore, the number of weights usually grows quite large as well\(^1\), which necessitates a “2D” coding scheme which codes both the data and the derivatives.

\(^1\)As a matter of fact it grows proportionately with the number of features or dimension of the dataset.
1.1. Related Work

The two algorithms which we use to benchmark our algorithm are Gradient Coding (Tandon et al., 2017) and K-Asynchronous Gradient Descent (Dutta et al., 2018b; 2021); however, many of the design of our coding scheme is also influenced by the works in (Lee et al., 2018), (Yu et al., 2017), and (Dutta et al., 2020).

1.1.1. Gradient Coding

In the gradient coding Gradient Coding (Tandon et al., 2017) scheme the main idea is to encode the derivatives with respect to the data partitions \( \frac{\partial}{\partial x} \) from Eq. (1); since the loss function in Eq. (2) splits up into a sum of smaller loss functions, \( l_i \), in terms of the partitions, \( D_i \), linear codes can be efficiently applied to the gradients \( \frac{\partial}{\partial y} l_i \). This work has gone on to spawn many works (Atallah & Rahnavard, 2018; Charles & Papailiopoulos, 2020; Ye & Abbe, 2018; Halbawi et al., 2018; Ozfatura et al., 2019b; Karakus et al., 2019; Ozfatura et al., 2019a; Horii et al., 2019; Raviv et al., 2020; Reisizadeh et al., 2019a; Maity et al., 2019; Bitar et al., 2019; Amiri & Gündüz, 2019; Reisizadeh et al., 2019b; Sasi et al., 2020; Wang et al., 2021; Ozfatura et al., 2020; Bitar et al., 2020; Zhang & Simeone, 2021); gradient coding is currently a vibrant topic of research. The main improvements of our coding scheme over the state of the art in Gradient Coding is that our code: can perform asynchronous coded updates, allows the backpropagation itself to be coded (which greatly reduces the communication complexity for high dimensional data), our code has 0 encoding and decoding overhead in terms of multiplications, and has an overall reduction in the redundancy of data/memory overhead.

1.1.2. Asynchronous Gradient Descent

The main idea in Asynchronous Gradient Descent (Ferdinand et al., 2017; Dutta et al., 2018b; Ferdinand & Draper, 2018; Ferdinand et al., 2020; Dutta et al., 2021) is to simply perform a gradient update whenever a specified number, \( k \), workers have returned. The name “asynchronous” comes from the eponymous concept in distributed computing where communication rounds are not synchronized.

1.1.3. General Coded Distributed Function Computation Schemes

The main idea in (Lee et al., 2018), (Yu et al., 2017), and (Dutta et al., 2020) is that one can distribute large matrix multiplications amongst workers and encode the smaller block matrix operations. The works initiated much research in distributed coded matrix multiplication (Dutta et al., 2020; Lee et al., 2017; Baharav et al., 2018; Dutta et al., 2018a; Wang et al., 2018; Soto et al., 2019; Dutta et al., 2019; Das & Ramamoorthy, 2019; Hong et al., 2021). Further work has been extended to include batch matrix multiplication as well (Yu et al., 2019; Jia & Jafar, 2020). The main drawback of these (multi-)linear methods is the non-linear activation functions; in particular, these methods can only encode the linear computations between the layers of a network. Another interesting approach is to attempt to encode the neural network itself (Kosai et al., 2018; 2019a;b); however, this approach suffers from long training times due to combinatorial explosion of different fault patterns is there are enough stragglers.

1.2. Contribution

The main contributions of this paper are to introduce a novel coding scheme for gradient descent that: allows for asynchronous gradient updates, maximizes the amount of information contained by random subsets of vectors, minimizes the weight of the code, compresses the gradient in a manner that scales well with the number of nodes, and achieves a lower a communication complexity and memory (storage) overhead with respect to the state of the art. Another improvement of our algorithm over the state of the art is to consider the correct information metric; all of the other coding schemes assume that the Hamming distance is the correct metric, which does not consider the natural (differential) geometry of the gradient. We will show that the correct distance is the one given by the real projective space \(^2\mathbb{RP}^n\), i.e., the Kähler metric. Furthermore, we will show that our coding scheme, i.e., our choice of coefficients, maximizes the amount of information returned by the workers and furthermore has zero decoding overhead (in terms of multiplications) since the master can just directly add and subtract the results returned by the workers without needing to decode the information.

1.3. Background

We quickly give some important definitions and background from coding theory, information theory, and geometry. In coded distributed computing an erasure code is a pair of functions \( \mathcal{C} = (\mathcal{E}, \mathcal{D}) \) where the workers tasks are given by the encoding procedure

\[
\{ \hat{\theta}_0, ..., \hat{\theta}_{n-1} \} := \mathcal{E} \{ \theta_0, ..., \theta_k \}
\]

and a decoding procedure for some family of fault-tolerant subsets, \( \mathcal{F}_{\mathcal{C}} \), such that

\[
\{ \hat{\theta}_1, ..., \hat{\theta}_{r} \} \in \mathcal{F}_{\mathcal{C}} \quad \Longrightarrow \quad \mathcal{D} \{ \hat{\theta}_1, ..., \hat{\theta}_{r} \} = \{ \theta_0, ..., \theta_k \}.
\]

If \( \mathcal{F}_{\mathcal{C}} \) consists of all the \( r \)-subsets (for some integer \( r \)) of \( \{ \hat{\theta}_0, ..., \hat{\theta}_{n-1} \} \), then \( \mathcal{C} \) can correct any \( r \) erasures or stragglers; furthermore, if \( r = n - k \) then the code is a maximum

\(^2\mathbb{RP}^n\) is defined as the set of all vectors in \( \mathbb{R}^{n+1} \) quotiented by the equivalence relation \( v \sim w \iff (\exists \lambda) \; v = \lambda w \).
distance separable (MDS) code. If the encoder \( E \) is given by a generator matrix \( G_C \), i.e., if

\[
E \begin{bmatrix} \theta_0 & \ldots & \theta_k \end{bmatrix}^T = G_C \begin{bmatrix} \theta_0 & \ldots & \theta_k \end{bmatrix}^T
\]

then \( C \) is called a linear code. The weight of a linear code is the maximum number of 0's in the rows of the matrix \( G_C \); the importance of the weight metric stems from the fact that it measures the amount of work that the workers do since the rows of \( G_C \) are the worker tasks \( \bar{t}_i \). Thus, in order to avoid confusion we will use \( t \) to denote the weight of the code as well as the number of tasks that each worker does; equivalently \( t \) is the number of data partitions on the workers. To further simplify notation we abuse notation and use \( C \) in place of \( G_C \) and \( E_C \) when the context is clear.

A potential point of confusion is that the \( \theta \) need not be the weights \( w \) of the \( b_w \). This is because the derivative of loss function \( L \) also implicitly takes the data \( D \) as an input; this is key insight in all gradient coding algorithms. One of the key insights of this paper is to allow the coded gradient to be linear combinations of both \( \frac{\partial}{\partial \theta_0} \) and the \( \frac{\partial}{\partial \theta_k} \). An important notational convention is that we let the \( D_i \) be partitions (or batches) of the data set instead of just datapoints as is common in the gradient coding literature, is particular \( D_0, \ldots, D_{t-1} \) denotes a partitioning of the data-set into \( t \) pieces.

The reason for the name “maximum distance separable code” is that an MDS maximizes the distances between the codewords \( E \{ \theta_0, \ldots, \theta_k \} \) using the Hamming distance; in particular, maximum distance separable means that the code words \( \bar{\theta} \in C \) have achieve the maximum \( \max_{C: \text{code}} \min_{\bar{\theta} \in C} d(\bar{\theta}, \bar{\theta}') \) where \( d \) is the Hamming distance. There are two problems with this approach: the first is that MDS codes in this context require arbitrarily large amount of work, i.e., they have a large weight, and the second is that the classical discrete MDS codes are using the wrong metric. This paper proposes to use the metric given by the projective geometry\(^3\) on the space of derivatives. Here we mean maximum distance separable with respect to the distance function \( d(\theta, \theta') = \min \{ \arccos(\bar{\theta}, \bar{\theta}''), \arccos(-\bar{\theta}, \bar{\theta}) \} \).

2. General Overview of the Design Principles

Consider the case where there are two derivatives and we wish to create two parity tasks using only summation and subtraction in the encoding procedure. Such a code is given by the following generator matrix

\[
C = \begin{bmatrix} \hat{\theta}_0 & \hat{\theta}_1 & \hat{\theta}_2 & \hat{\theta}_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}
\]

which adds fault tolerance to the job \( I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \) with the parity tasks \( P = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \). This code has the serendipitous property of having negligible decoding complexity and negligible communication complexity! For example, if the master receives \( \nabla \) in the direction \( \bar{\theta}_3 = \frac{1}{\sqrt{2}}(\theta_0 + \theta_1) \), then the master can decrease both \( \theta_0 \) and \( \theta_1 \) by the value returned by \( \mathcal{W}_2 \), i.e., \( \bar{\theta}_2 \), if the master receives \( \nabla \) in the direction \( \bar{\theta}_2 = \frac{1}{\sqrt{2}}(\theta_0 - \theta_1) \), then the master can decrease \( \theta_0 \) and increase \( \theta_1 \) by the value returned by \( \mathcal{W}_3 \), i.e., \( \bar{\theta}_3 \). The master need only perform 2 additions/subtractions, and more generally (see Eq. 4) if there are \( t \) “sub-tasks” the master only needs to perform \( t \) additions/subtractions. The multiplication by \( \frac{1}{\sqrt{2}} \) can be subsumed by the learning rate; thus, our code has zero multiplication overhead. Furthermore, this information can be communicated using only one float, since the master knows which direction/worker the derivative was computed from.

2.1. What is the Optimal Choice of Directions?

Looking at Fig. 1, we see that the code \( G \) is MDS in the sense that it maximizes the independence between the vectors. Equivalently \(^4\) \( G \) minimizes the confusion between codewords or minimizes the mutual information between codewords; thereby maximizing the entropy or the information content. As we will soon see this has the effect of allowing lossy low-distortion compression for larger codes. A second contribution of this paper is to show how to preserve an approximate MDS property for larger codes which allows for this form of compression.

In what sense does \( C \) being MDS imply fault tolerance? The following example illustrates one kind of error which the code is immune to:

Consider the case where two workers return the derivatives in the directions \( \bar{\theta}' = \left( \frac{\pi}{4} - \epsilon, \frac{\pi}{4} + \epsilon \right) \). By inspecting the second diagram in Fig. 1, it is easy to see that \( \lim_{\epsilon \to 0} \bar{\theta}' \to -\bar{\theta} \), so that \( \lim_{\epsilon \to 0} \arccos \left( \bar{\theta}, \bar{\theta}' \right) \to \pi \). We will also show that the

\(^3\)See (Kühnel et al., 2006) for the case \( \mathbb{R}P^2 \) and Appx. 3 of (Vogtmann et al., 2013) or Thm. 10.2 in ch. 3 of (Suetin et al., 1989) for the more general case \( \mathbb{C}P^n \).

\(^4\)Under the assumption that the data \( D_i \) are i.i.d. See (Cover & Thomas, 2006) for why maximal entropy maximizes information sent through a message.
The last example did not allow us to show the more general very special case. Therefore before showing the general and that error in the derivative can get bigger and bigger as $\epsilon \to 0$. If worker one computes $\frac{\partial C}{\partial \theta_0}$, worker two computes $\frac{\partial C}{\partial \theta_1}$, and $\frac{\partial C}{\partial \theta_i} = (-1)^i L$, then we have that $\frac{\partial C}{\partial \theta_0} = \frac{1}{\sqrt{2}}(L + (-L)) = 0$ and that $\frac{\partial C}{\partial \theta_1} = \cos \left(\frac{\pi}{4} - \epsilon\right) L - \sin \left(\frac{\pi}{4} - \epsilon\right) L = \left(\cos \left(\frac{\pi}{4} - \epsilon\right) - \sin \left(\frac{\pi}{4} - \epsilon\right)\right) L$. Therefore if both ($\epsilon \approx 0$) and ($L >> 1$), then $\frac{\partial C}{\partial \theta_0} \approx \frac{\partial C}{\partial \theta_1} \approx 0$, which is an error; when the master receives the messages from workers one and two she will think she has arrived at a optimal fit since both $\frac{\partial C}{\partial \theta_0}$ and $\frac{\partial C}{\partial \theta_1}$ are very small; i.e., the master may halt the algorithm on a terrible fit. Furthermore it is easy to see that $\max_{\epsilon} \frac{\partial C}{\partial \theta_i}$ occurs when $\epsilon = \frac{\pi}{4}$, i.e., when $\arccos \left(\frac{\theta}{\sqrt{2}}\right) = \frac{\pi}{2}$ so that our previous choice is optimal.

2.2. Motivating Example: Base Case of LPD Codes

The last example did not allow us to show the more general lossy compression phenomenon that can occur for more general codes. Also we will soon prove that it is impossible to have MDS codes for large dimensions where the workers perform a small amount of work\textsuperscript{3}. In a sense $k = 2$ is a very special case. Therefore before showing the general compression phenomenon let us show how to “compress the derivative” for $k = 4$.

\textsuperscript{3}This follows from a general rule of thumb in coding theory which states that a code cannot simultaneously have a sparse matrix and be MDS; however we will prove it rigorously for our case.

### Figure 1

The code $C$ is MDS with respect to the the distance between the codewords. It is easy to see that $-\theta_i$ carries the same information as $\theta_i$ and it is therefore inefficient to include it since it is a form of replication coding. Likewise we should not include a vector that makes a small angle with $\theta_i$ for the same reason. The geometry of this problem is that of $\mathbb{R}^p$, i.e., the real protective line. This is because the directions $\theta$ and $-\theta$ are information theoretically equivalent. The second figure displays the relationship between $\theta$ and $\theta'$. As $\theta'$ becomes more linearly independent from $\theta$ it begins to carry more novel information that cannot be inferred from $\theta$. At the “greenest” extremes $\theta$ and $\theta'$ become statistically independent which is the maximum entropy configuration. Maximum entropy is equivalent (see Fn. 4) to maximum information about the $\theta_i$.

Suppose that we have 8 workers $\tilde{\theta}_i$, loss function $l(h, y) = \frac{1}{2}[|h - y|^2]$, $u$ input features $x_i$, and space of hypothesis functions

$$\mathcal{H} = \left\{ h_w = (y_1, ..., y_u) \mid y_i = \frac{e^{w_j^T x_i}}{1 + \sum_j w_j^T x_i}, w_i \in \mathbb{R}^2 \right\},$$

where $w_j^T x = w_{j,1} x_1 + ... + w_{j,u} x_u$; i.e., $\mathcal{H}$ is the space of multinomial logistic regression functions (however, this procedure will work for any feed-forward neural network, see Fig. 2). Similarly to the previous design we can give the following directions to the workers

$$C^{(8,4,2)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}; \quad (3)$$

however, this time we let $\frac{\partial}{\partial \theta_i} =$ “the derivative of the first half of the output nodes with respect to the first half of the dataset”, $\frac{\partial}{\partial \theta_i} =$ “the derivative of the second half of the output nodes with respect to the first half of the dataset”, $\frac{\partial}{\partial \theta_i} =$ “the derivative of the first half of the output nodes with respect to the second half of the dataset”, and $\frac{\partial}{\partial \theta_i} =$ “the derivative of the second half of the output nodes with respect to the second half of the dataset”. We can see that one way that there is lossy compression is that the 4 workers don’t necessarily return the gradient perfectly, but we will later prove that they return a pretty good approximation of it. However, we had a second further lossy compression step to our code; we give workers 5 and 6 the approximation of it. However, we had a second further lossy compression step to our code; we give workers 7 and 8 the data partitions $D_2, D_3$ and give workers 7 and 8 the data partitions $D_1, D_4$.

3. General Code Construction

We first show how to construct what we will denote as a $[n, k, t]$-projective derivative code, or $[n, k, t]$-code, for $n = 2^m$, $k = 2^p$ and $t = 2^q$, i.e., $n, k, t$ are all powers of 2, and then show how to use “cyclic” and “toroidal” permutations to construct the code for more general $k, n$; however the $t$ is always chosen to be a power of two for reasons that will soon become clear. The parameter $n$ is the number of workers, $k$ the number of derivatives/features, and $t$ is the number of sub-tasks that each worker will perform; i.e., the number of derivatives per worker.
3.1. The Characteristic Vectors

To construct the code we first construct the characteristic vectors from the following family of functions $\chi_\alpha : \mathbb{F}_2^p \to \mathbb{C}$, defined by the lambda expression

$$\chi_\alpha : \beta \mapsto \frac{1}{\sqrt{2^p}} e^{(\alpha,\beta)\pi i} = \frac{(-1)^{(\alpha,\beta)}}{\sqrt{2^p}},$$

where $\alpha, \beta \in \mathbb{F}_2^p$ are defined as binary strings of length $t$ and $\langle \alpha, \beta \rangle$ is the dot product on $\alpha, \beta \in \mathbb{F}_2^p$, which is equivalent to taking the bit-wise AND\(^6\) of $\alpha$ and $\beta$ and then taking the XOR\(^7\) of the result. In particular, $\langle \alpha, \beta \rangle$ is defined as $\langle \alpha, \beta \rangle = (\alpha_0 \land \beta_0) \oplus (\alpha_1 \land \beta_1) \oplus \cdots \oplus (\alpha_{t-1} \land \beta_{t-1})$, where $\oplus$ is as defined in fn. 7.

It is an elementary fact of representation theory that the vectors, $\chi_\alpha$, correspond to the irreducible representations of $\mathbb{F}_2^p$ in $\mathbb{C}$ and are therefore an orthogonal basis see Thm. 6 of (Serre, 2012) or Thm. 2.12 of (Fulton & Harris, 1991). One can also prove this fact by direct computation using discrete Fourier analysis see ch. 4 of (Tao & Vu, 2006).

These functions are well-studied in discrete mathematics and usually referred to as the (additive) characters of $\mathbb{F}_2^p$.

Let us construct these vectors for $p = 2$ and verify the veracity of these statements for that case. The binary strings of length 2 are $\alpha \in \{00, 01, 10, 11\}$ and this corresponds to the functions

$$X^4 = \left[ \begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \\ \chi_{00} & \chi_{01} & \chi_{10} & \chi_{11} \end{array} \right] = \left[ \begin{array}{cccc} -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -2 & -2 & -2 & -2 \end{array} \right] e^{(\alpha,\beta)\pi i}$$

where $p = \log(t)$ and $\chi_\alpha : \mathbb{F}_2^p \to \mathbb{C}$, let $L^{(t)}$ and $R^{(t)}$ be the matrices defined by the equation

$$L^{(2^t)} = \begin{bmatrix} 0^{(t)} & \frac{1}{\sqrt{2}} X^{(t)} \\ 0^{(t)} & \frac{1}{\sqrt{2}} X^{(t)} \end{bmatrix}, \quad R^{(2^t)} = \begin{bmatrix} \frac{1}{\sqrt{2}} X^{(t)} & 0^{(t)} \\ \frac{1}{\sqrt{2}} X^{(t)} & 0^{(t)} \end{bmatrix},$$

then we can define $C^{(2k, k, t)}$, the generator for the $[2k, k, t]$-code, as

$$C^{(2k, k, t)} = \begin{bmatrix} \chi_{00} & \chi_{01} & \chi_{10} & \chi_{11} \\ \theta_0^{(t)} & \theta_1^{(t)} & \theta_2^{(t)} & \theta_3^{(t)} \end{bmatrix}$$

\(^6\)AND is the logical conjunction, denoted by $\land$, i.e., $a \land b = 1$ if $a = b = 1$ and $a \land b = 0$ otherwise. The bit-wise AND of two sequences $\alpha = \alpha_0 \ldots \alpha_{t-1}, \beta = \beta_0 \ldots \beta_{t-1}$ is $\alpha \land \beta = (\alpha_0 \land \beta_0) (\alpha_1 \land \beta_1) \ldots (\alpha_{t-1} \land \beta_{t-1})$.

\(^7\)XOR is the exclusive or, denoted by $\oplus$, i.e., $a \oplus b = 1$ if $a \neq b$ and $a \oplus b = 0$ if $a = b$. Figure 2. The different ways to partition the backpropagation gradient. The first partition shows how to partition the gradient for a simple neural network with no hidden nodes. The other partition corresponds to a more general deep-neural network with hidden nodes. The last partition shows how to apply the recursive step in Alg. 2.
where $\theta$ is the ratio of tasks to sub-tasks, $\theta_i^{(t)}$ is the sequence of sub-tasks $\theta_{it}$ through $\theta_{(i+1)t-1}$, and $\theta_i^{(t)}$ is similarly defined as a sequence of $\theta$ consecutive workers. Equivalently if we define the “rectangles” $R_{u,v}^{(t)}$ as

$$R_{u,v}^{(t)} = \{(i,j) \in \mathbb{N}^2 \mid ut \leq i < (u+1)t, vt \leq j < (v+1)t\},$$

then we can define $C^{(2k,t)}$ coordinate-wise as

$$C^{(2k,t)}_{\theta_i,\theta_j} = \begin{cases} X^{(t)}_{v_\theta t,j} \text{ if } i < k \text{ and } (i,j) \in R_{u,v}^{(t)} \cap [t] \times [t] + \frac{1}{2} \times [t] & \text{if } k \leq i \text{ and } (i,j) \in R_{u,v}^{(t)} \cap [t] + \frac{1}{2} \times [t] \text{ and } t \leq i \text{ and } (i,j) \in R_{u,v}^{(t)} \cap [t] + \frac{1}{2} \times [t] \text{ or } k \leq i \\ 0 \text{ otherwise.} \end{cases}$$

It is straightforward to prove the following beautiful property property

**Lemma 3.1.** The matrices $X^{(t)}$ satisfy the following recursion relation $X^{(2t)} = X^{(2)} \otimes X^{(t)}$.

An alternative is the weaker statement $X^{(2)}$ is a Hadamard matrix and the tensor product of two Hadamard matrices is a Hadamard matrix” whose proof can be found in (Huffman & Pless, 2003).

### 3.2.1. Data-Gradient-Partition for Powers of Two

Similar to the example given in Sec. 2.2 we give the workers $\theta_i$ the data partition given by Alg. 1. The idea behind Alg. 1 is simple; we give the first worker $Data[\theta_0] = D_0, \ldots, D_{t-1}$, and the second worker $Data[\theta_1] = D_t, \ldots, D_{2t-1}$, and so on until worker $k$, at which point we give the workers $k, \ldots, n$ a cyclic shift of the previous assignment, e.g. worker $k$ gets $Data[\theta_k] = \sum (D_{i} \ldots D_{i+t-1})$.

The procedure for partitioning and encoding the gradients, Alg. 2, is slightly more involved; however the main idea is illustrated in Fig. 2. The main idea behind Alg. 2 is encode the gradient in the manner in which backpropagation occurs; this allows for the iterative decoding/gradient update at the master node which allows for asynchronous gradient updating.

### 3.3. Construction for General Parameters

Given some general $(n,k,t)$ we use construct the matrix $C^{(n,k,t)}$, where $n'$ and $k'$ are the next nearest powers of 2 (repeating rows if necessary) and use a “2-D” permutation algorithm similar to (Fan et al., 2020) to distribute the sub-tasks in each round; however our algorithm uses more general (prime number) step-sizes chosen in each round and the permutations now occur in “higher dimensions.$^{10}$” In particular; we now use a similar procedure to permute tasks amongst workers if $n$ and $k$ are not powers of 2. For example if we have $n = 6$ workers and $k = 3$ tasks we can add extra “virtual tasks $\theta_3, \theta_4, \theta_5, \ldots, \theta_x = \theta_x \ldots \theta_x \ldots \theta_x \ldots \theta_x$ and perform the following “torodial” permutations on $C^{(5,3,2)}$ so that at round $r$ worker $i$ performs task $\theta_{i+r \mod n'}$ and similarly at round $r$ we have $t_i = \theta_{i+r \mod k}$.
More generally we find a displacement $d$ equal to an (odd) prime number that is co-prime\footnote{Although it is notoriously hard to find a prime divisor of number, it is surprisingly easy to find a prime non-divisor. This easy to see since one can just test divisibility by 2,3,5,... and since the product of the first primes less than 100 is approximately equal to $2^{100}$ this will halt very quickly, i.e. it will halt in less than 25 steps for $k < 2^{100}$ since there are only 25 primes less than 100.} to $k$ and we let worker $i$ performs task $\theta_i + dz \% n$ at round $r$ and let $t_i = \theta_i + r \% k$ at round $r$. This allows gives the following statistical “uniformity lemma.”

**Lemma 3.2.** If the displacement, $d$, is equal to an (odd) prime number that is co-prime to $k$ then the blue rectangle in (the general form of) Eq. 5 will visit every entry in the matrix with every possible pattern of $X(t)$ and every cyclic permutation of the $t_i$ contained inside of the blue rectangle.

4. Analysis and Evaluation

4.1. Theoretical Analysis

In this section we give a theoretical comparison of the algorithms, see Table 1, and we prove theorems regarding the existence and non-existence of codes with certain properties. The following theorem, i.e., Thm. 4.1, shows that Hamming-distance MDS coding schemes must have the workers do an arbitrarily large amount of work. We then later show that codes that are approximately MDS with respect to the projective geometry metric are maximized the amount of information sent back by the workers while keeping the amount of work done by the workers as low as possible; i.e., there are approximately projective-MDS that have weights $t = 2, ..., n$.

**Theorem 4.1.** If the parameters $(n, k, t)$ satisfy $t \leq n - k$ then there is no Hamming-distance MDS $(n, k)$-code for the derivatives.

In particular; the proof of Thm. 4.1 can be strengthened to say that:

**Corollary 4.2.** In an MDS $(n, k)$-coding scheme $A(C)_i = 0$, for $i \leq n - k$.

The importance of Cor. 4.2 is made clear through the following interpretation:

**Corollary 4.3.** In an MDS $(n, k)$-coding scheme all of the workers must do at least $n - k$ amount of work.

However a simple observation of the construction given in Sec. 3.2 gives us that:

**Theorem 4.4.** There exists $(n, k, t)$-LPD codes for any $t \geq 2$.

The next theorem proves that under the projective distance we have that our code achieves approximately maximal distance.

**Theorem 4.5.** The family $(n, k, t)$-code are approximately MDS $(n, k)$-code for the derivatives in the projective-distance for $n \leq 2k$.

The proof of Thm. 4.5 gives us that the distance between any two codewords $\theta, \bar{\theta}$ is bounded above by $d(C) = \min_{\theta, \bar{\theta} \in C} d(\theta, \bar{\theta}) = \frac{2}{3}$ and thus in term of percentages of the optimal $\frac{d}{2}$ we have

$$\frac{d}{2} - d(C) = \frac{5}{6} \approx 83\%$$

of the “theoretical” optimal distance; however there can be no code that achieves the “theoretical” optimal distance:

**Theorem 4.6.** The percentage in Eq. 6 cannot be made 100%; i.e., there are no projective MDS codes for $n > k$.

4.2. Experimental Results

The experiments were run on AWS Spot Instances; the workers are AWS EC2 c5a.large instances (compute optimized)
Table 1. Comparison of main algorithms.

| Code Scheme | Encoding Complexity | Communication Complexity | Decoding Complexity | Weight Range | Asynchronous? | Parameter Compression? |
|-------------|---------------------|--------------------------|---------------------|--------------|--------------|------------------------|
| LPDC        | 0                   | $O(\frac{k}{t})$         | 0                   | $t \in [2, \frac{n}{2}]$ | √            | √                     |
| GC          | $O(nk)$             | $O(k)$                   | $O(k^\omega)$       | $t = n - k + 1$ | ×            | ×                     |
| K-AC        | 0                   | $O(k)$                   | 0                   | $t \in [1, n]$ | √            | ×                     |

and master is an AWS EC2 r3.large instance (Memory Optimized). The experimental procedure was written using Mpi4py (Dalcín et al., 2005; 2008) in Python. We used a modification of the code in (Tandon, 2017) written by the first author of (Tandon et al., 2017) to implement Gradient Coding (GC) as well as the random data generation; the implementation only supported logistic regression and we generalized it to support multinomial logistic regression (i.e., more than one class). The software in (Tandon, 2017) used a Gaussian mixture model of two distributions to create input features for the logistic model; we generalized it to allow for an arbitrary number of Gaussian distributions in the mixture to create a robust data set. To be as fair as possible in our comparison with K-Asynchronous Gradient descent (K-AC) (Dutta et al., 2018b) we made setup for K-AC nearly identical with the exception of the coding scheme; i.e., K-AC and LPD used the exact same data partitions and same number of $k$ workers in the $k$-asynchronous batches.

We ran experiments with 8 workers (see Fig. 4), 16 workers (see Fig. 4), and 32 workers (see Fig. 4). The testing error (i.e., the workers do not train on the test data) is plotted against the time. In all of the experiments we ran LPD codes converged far faster; however, it often overfitted and sometimes the other algorithms would eventually get a lower test error. There are two possible explanations for this: either LPDC converges so much faster than the other algorithms that they never get a chance to overfit or the noise that initially helps LPDC find a very quick solution eventually causes it to stay some distance from the optimal solution. There is evidence for both of these possibilities because there are experiments were the other algorithms do not catch up to LPDC; see Sec. B for more results.

5. Conclusion and Open Problems

We propose LPDC codes which allow for asynchronous gradient updates by maximizing the amount of information contained by random subsets of vectors and minimizing the weight of the code. Our code compresses the gradient in a manner that scales well with the number of nodes (and the dimension of the data) and achieves a lower a communication complexity and memory overhead with respect to the state of the art. Another improvement of our algorithm over the state of the art is our discovery of the correct information metric; all of the other coding schemes assume that the Hamming distance is the correct metric, which does not consider the natural (differential) geometry of the gradient. Furthermore, we showed that our code was very efficient since the master can just directly add and subtract the results returned by the workers without needing to decode the information. We proved many of the complexity guarantees theoretically and also provided much empirical evidence for the performance. For future work we would like to strengthen the theoretical results by proving stronger complexity bounds as well as further investigating the effect of noise (or lossy compression) on the performance; it seems that at first the lossy compression is a great help but eventually it causes over fitting.

References

Amiri, M. M. and Gündüz, D. Computation scheduling for distributed machine learning with straggling workers. In
Lightweight Projective Derivative Codes

Dutta, S., Joshi, G., Ghosh, S., Dube, P., and Nagpurkar, P. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In Storkey, A. and Perez-Cruz, F. (eds.), Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, volume 84 of Proceedings of Machine Learning Research, pp. 803–812. PMLR, 09–11 Apr 2018b.

Dutta, S., Cadambe, V., and Grover, P. “short-dot”: Computing large linear transforms distributedly using coded short dot products. *IEEE Transactions on Information Theory*, 65(10):6171–6193, 2019. doi: 10.1109/TIT.2019.2927558.

Dutta, S., Joshi, G., Ghosh, S., Dube, P., and Nagpurkar, P. Slow and stale gradients can win the race. *IEEE Journal on Selected Areas in Information Theory*, 2:1012–1024, 2021.

Fan, X., Soto, P., Zhong, X., Xi, D., Wang, Y., and Li, J. Leveraging stragglers in coded computing with heterogeneous servers. In 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), pp. 1–10, 2020. doi: 10.1109/IWQoS49365.2020.9213028.

Ferdinand, N. S. and Draper, S. C. Anytime stochastic gradient descent: A time to hear from all the workers. In 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 552–559, 2018.

Ferdinand, N. S., Gharachorloo, B., and Draper, S. C. Anytime exploitation of stragglers in synchronous stochastic gradient descent. 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 141–146, 2017.

Ferdinand, N. S., Al-Lawati, H., Draper, S. C., and Nokleby, M. S. Anytime minibatch: Exploiting stragglers in online distributed optimization. *CoRR*, abs/2006.05752, 2020.
Lightweight Projective Derivative Codes

Halbawi, W., Azizan, N., Salehi, F., and Hassibi, B. Improving distributed gradient descent using reed-solomon codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 2027–2031, 2018. doi: 10.1109/ISIT.2018.8437467.

Hong, S., Yang, H., Yoon, Y., Cho, T., and Lee, J. Chebyshev polynomial codes: Task entanglement-based coding for distributed matrix multiplication. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4319–4327. PMLR, 18–24 Jul 2021.

Horii, S., Yoshida, T., Kobayashi, M., and Matsushima, T. Distributed stochastic gradient descent using ldgm codes. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1417–1421, 2019. doi: 10.1109/ISIT.2019.8849580.

Huffman, W. and Pless, V. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003. ISBN 9780521782807. URL https://books.google.com/books?id=B2FjPXtS_QUC.

Ireland, K. and Rosen, M. *A classical introduction to modern number theory*. Graduate texts in mathematics. Springer-Verlag, 1982. ISBN 9783540906254. URL https://books.google.com/books?id=WvjuAAAAMAAJ.

Jia, Z. and Jafar, S. A. Generalized cross subspace alignment codes for coded distributed batch matrix multiplication. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020. doi: 10.1109/ICC40277.2020.9149322.

Karakuš, C., Sun, Y., Diggavi, S., and Yin, W. Redundancy strategies for straggler mitigation in distributed optimization and learning. *Journal of Machine Learning Research*, 20(72):1–47, 2019.

Kosaian, J., Rashmi, K. V., and Venkataraman, S. Learning a code: Machine learning for approximate non-linear coded computation. *CoRR*, abs/1806.01259, 2018.

Kosaian, J., Rashmi, K. V., and Venkataraman, S. Parity models: Erasure-coded resilience for prediction serving systems. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP ’19, pp. 30–46, New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 9781450368735. doi: 10.1145/3341301.3359654.

Kosaian, J., Rashmi, K. V., and Venkataraman, S. Parity models: A general framework for coding-based resilience in ml inference. *ArXiv*, abs/1905.00863, 2019b.

Kühnel, W., Hunt, B., and Society, A. M. *Differential Geometry: Curves - Surfaces - Manifolds*. Student mathematical library. American Mathematical Society, 2006. ISBN 9780821839881. URL https://books.google.com/books?id=TyqUnlyV4Y4C.

Lee, K., Suh, C., and Ramchandran, K. High-dimensional coded matrix multiplication. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2418–2422, 2017. doi: 10.1109/ISIT.2017.8006963.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding Up Distributed Machine Learning Using Codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2018.

Maity, R. K., Rawa, A. S., and Mazumdar, A. Robust gradient descent via moment encoding and ldpc codes. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 2734–2738. IEEE, 2019.

Ozfatura, E., Gündüz, D., and Ulukus, S. Gradient coding with clustering and multi-message communication. In *2019 IEEE Data Science Workshop (DSW)*, pp. 42–46. IEEE, 2019a.

Ozfatura, E., Gündüz, D., and Ulukus, S. Speeding up distributed gradient descent by utilizing non-persistent stragglers. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 2729–2733, 2019b. doi: 10.1109/ISIT.2019.8849684.

Ozfatura, E., Ulukus, S., and Gündüz, D. Distributed gradient descent with coded partial gradient computations. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3492–3496, 2019c.

Ozfatura, E., Ulukus, S., and Gündüz, D. Straggler-aware distributed learning: Communication–computation latency trade-off. *Entropy*, 22(5), 2020. ISSN 1099-4300. doi: 10.3390/e22050544. URL https://www.mdpi.com/1099-4300/22/5/544.

Raviv, N., Tamo, I., Tandon, R., and Dimakis, A. G. Gradient coding from cyclic mds codes and expander graphs. *IEEE Transactions on Information Theory*, 66(12):7475–7489, 2020. doi: 10.1109/TIT.2020.3029396.

Reisizadeh, A., Prakash, S., Pedarsani, R., and Avestimehr, A. S. Tree gradient coding. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 2808–2812, 2019a. doi: 10.1109/ISIT.2019.8849431.

Reisizadeh, A., Taheri, H., Mokhtari, A., Hassani, H., and Pedarsani, R. Robust and communication-efficient collaborative learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R. (eds.),
Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019b.

Sasi, S., Lalitha, V., Aggarwal, V., and Rajan, B. S. Straggler mitigation with tiered gradient codes. *IEEE Transactions on Communications*, 68(8):4632–4647, 2020. doi: 10.1109/TCOMM.2020.2992721.

Serre, J. *Linear Representations of Finite Groups*. Graduate Texts in Mathematics. Springer New York, 2012. ISBN 9781468494587. URL https://books.google.com/books?id=9mT1BwAAQBAJ.

Soto, P., Li, J., and Fan, X. Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5937–5945. PMLR, 09–15 Jun 2019.

Suetin, P. K., Kostrikin, A. I., and Manin, Y. I. Linear algebra and geometry. 1989.

Tandon, R. gradient coding, 2017.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient coding: Avoiding stragglers in distributed learning. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3368–3376. PMLR, 06–11 Aug 2017.

Tao, T. and Vu, V. *Additive Combinatorics*. Number v. 13 in Additive combinatorics. Cambridge University Press, 2006. ISBN 9780521853866. URL https://books.google.com/books?id=WBXYzwEACAAJ.

Vogtmann, K., Weinstein, A., and Arnol’d, V. *Mathematical Methods of Classical Mechanics*. Graduate Texts in Mathematics. Springer New York, 2013. ISBN 9781475720648. URL https://books.google.com/books?id=BXIAswEACAAJ.

Wang, H., Guo, S., Tang, B., Li, R., Yang, Y., Qu, Z., and Wang, Y. Heterogeneity-aware gradient coding for tolerating and leveraging stragglers. *IEEE Transactions on Computers*, pp. 1–1, 2021. doi: 10.1109/TC.2021.3063180.

Wang, S., Liu, J., and Shroff, N. Coded sparse matrix multiplication. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5152–5160. PMLR, 10–15 Jul 2018.

Ye, M. and Abbe, E. Communication-computation efficient gradient coding. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5610–5619. PMLR, 10–15 Jul 2018.
A. Proofs of Theorems

Lemma A.1. The matrices $X^{(t)}$ satisfy the following recursion relation $X^{(2t)} = X^{(2)} \otimes X^{(t)}$

Proof. This is a direct consequence of Thm. 10 in (Serre, 2012).

Lemma A.2. If the displacement, $d$, is equal to an (odd) prime number that is co-prime to $k$ then the blue rectangle in (the general form of) Eq. 5 will visit every entry in the matrix with every possible pattern of $X^{(t)}$ and every cyclic permutation of the $t_i$ contained inside of the blue rectangle.

Proof. The leftmost point of the blue rectangle is equal to $(i, i+r(1, d) \equiv (i+r, i+dr) \mod \mathbb{Z}/n'\mathbb{Z} \times \mathbb{Z}/k\mathbb{Z}$. By the Chinese remainder theorem (see (Ireland & Rosen, 1982) or (Dummit & Foote, 2003)) $(1, d)$ is a generator of $\mod \mathbb{Z}/n'\mathbb{Z} \times \mathbb{Z}/k\mathbb{Z}$ since $d$ is coprime to $1$, $k$, and $n'$.

Theorem A.3. If the parameters $(n, k, t)$ satisfy $t \leq n - k$ then there is no Hamming-distance MDS $(n, k)$-code for the derivatives.

Proof. If $A(C)_i =$ “number of rows of weight $i$”, then Theorem 7.4.1 in (Huffman & Pless, 2003) gives us that an MDS will have $A(C)_i = 0$ for $i \leq n - k$.

Theorem A.4. The family $(n, k, t)$-code are approximately MDS $(n, k)$-code for the derivatives in the projective-distance for $n \leq 2k$.

Proof. By Lem. 3.2 it suffices to prove this for powers of two. The distance between any vectors is $\arccos \frac{1}{2} = \frac{\pi}{3}$ and this only happens for $t$ out of $n$ choices for any vector; the distance is equal to the maximum $\frac{\pi}{2}$ for all other vectors.

Theorem A.5. The percentage in Eq. 6 cannot be made $100\%$; i.e., there are no projective MDS codes for $n > k$.

Proof. If this statement was false there would be $n + 1$ linearly independant vectors in $n$-dimensional space, $\mathbb{F}^n$. 


B. Experimental Results

Figure 4. Experiments with 8 workers.
Figure 5. Experiments with 16 workers.
Figure 6. Experiments with 32 workers.