Natural Compression for Distributed Deep Learning

Samuel Horváth  
KAUST, KSA  
samuel.horvath@kaust.edu.sa

Chen-Yu Ho  
KAUST, KSA  
chenyu.ho@kaust.edu.sa

L'udovít Horváth  
Comenius University, Slovakia  
ludovit.horvath.94@gmail.com

Atal Narayan Sahu  
KAUST, KSA  
atal.sahu@kaust.edu.sa

Marco Canini  
KAUST, KSA  
marco@kaust.edu.sa

Peter Richtárik  
KAUST, KSA and MIPT, Russia  
peter.richtarik@kaust.edu.sa

Abstract

Due to their hunger for big data, modern deep learning models are trained in parallel, often in distributed environments, where communication of model updates is the bottleneck. Various update compression (e.g., quantization, sparsification, dithering) techniques \cite{2, 47, 48, 22} have been proposed in recent years as a successful tool to alleviate this problem. In this work, we introduce a new, remarkably simple and theoretically and practically effective compression technique, which we call natural compression ($C_{\text{nat}}$). Our technique is applied individually to all entries of the to-be-compressed update vector and works by randomized rounding to the nearest (negative or positive) power of two. $C_{\text{nat}}$ is “natural” since the nearest power of two of a real expressed as a float can be obtained without any computation, simply by ignoring the mantissa. We show that compared to no compression, $C_{\text{nat}}$ increases the second moment of the compressed vector by the tiny factor $\frac{9}{8}$ only, which means that the effect of $C_{\text{nat}}$ on the convergence speed of popular training algorithms, such as distributed SGD, is negligible. However, the communications savings enabled by $C_{\text{nat}}$ are substantial, leading to $3\times-4\times$ improvement in overall theoretical running time. For applications requiring more aggressive compression, we generalize $C_{\text{nat}}$ to natural dithering, which we prove is exponentially better than the immensely popular random dithering technique \cite{13, 39}. Our compression operators can be used on their own or in combination with existing operators for a more aggressive combined effect. Finally, we show that $C_{\text{nat}}$ is particularly effective for the in-network aggregation (INA) \cite{40} framework for distributed training, where the update aggregation is done on a switch, which can only perform integer computations.

1 Introduction

Modern deep learning models \cite{18} are almost invariably trained in parallel or distributed environments, which is necessitated by the enormous size of the data sets and dimension and complexity of the models required to obtain state-of-the-art performance. In our work, the focus is on the data-parallel paradigm, in which the training data is split across several workers capable of operating in
A key bottleneck of the above algorithm, and of its many variants (e.g., variants utilizing mini-batch scheduling), is the communication cost per iteration and the number of communication rounds. Remarkably, for carefully constructed compression strategies and appropriately designed training algorithms, this trade-off can be captured also theoretically.

\begin{equation}
\min_{x \in \mathbb{R}^d} f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x),
\end{equation}

where $x \in \mathbb{R}^d$ represents the parameters of the model, $n$ is the number of workers, and $f_i : \mathbb{R}^d \to \mathbb{R}$ is a loss function composed of data stored on worker $i$. Typically, $f_i$ is modelled as a function of the form $f_i(x) := E_{\zeta \sim D_i} [f_i(x)]$, where $D_i$ is the distribution of data stored on worker $i$, and $f_i : \mathbb{R}^d \to \mathbb{R}$ is the loss of model $x$ on data point $\zeta$. The distributions $D_1, \ldots, D_n$ can be different on every node, which means that the functions $f_1, \ldots, f_n$ may have different minimizers. This framework covers i) stochastic optimization when either $n = 1$ or all $D_i$ are identical, and ii) empirical risk minimization when $f_i(x)$ can be expressed as a finite average, i.e., $\frac{1}{m_i} \sum_{i=1}^{m_i} f_{ij}(x)$ for some $f_{ij} : \mathbb{R}^d \to \mathbb{R}$.

**Distributed Learning.** Typically, problem (1) is solved by distributed stochastic gradient descent (SGD) \cite{SGD}, which works as follows: i) given model $x^k$ maintained on each node, machine $i$ computes a random vector $g_i(x^k)$ whose mean is $\nabla f_i(x^k)$ (i.e., a stochastic gradient), ii) all stochastic gradients $g_i(x^k)$ are sent to a master node \footnote{There are several alternatives to this, all logically identical, but differing in the implementation. For instance, one may dispense off the master node and instead let all workers broadcast their gradient updates directly to their peers in an all to all fashion. Aggregation is then performed by each node separately. In the theoretical part of paper we work with an abstract method allowing for multiple specific implementations.}, \footnote{There are several alternatives to this, all logically identical, but differing in the implementation. For instance, one may dispense off the master node and instead let all workers broadcast their gradient updates directly to their peers in an all to all fashion. Aggregation is then performed by each node separately. In the theoretical part of paper we work with an abstract method allowing for multiple specific implementations.} which performs update aggregation $g^k = \sum_i g_i(x^k)$, iii) the aggregated gradient $g^k$ is sent back to the workers, and finally iv) all workers perform a single step of SGD: $x^{k+1} = x^k - \frac{\eta}{n} g^k$, where $\eta > 0$ is a step size. This iterative process is repeated until a model of suitable properties is found.

A key bottleneck of the above algorithm, and of its many variants (e.g., variants utilizing mini-batching \cite{DBLP:journals/corr/Ilios15}, importance sampling \cite{DBLP:journals/corr/Huang15, DBLP:journals/corr/Jacot17}, momentum \cite{DBLP:journals/corr/Sutskever13}, or variance reduction \cite{DBLP:journals/corr/Arjevani15, DBLP:journals/corr/Arjevani16}), is the communication cost of the typically dense gradient vector $g_i(x^k)$, and in a parameter-sever implementation with a master node, also the cost of broadcasting the aggregated gradient $g^k$. These are $d$ dimensional vectors of floats, with $d$ being very large in modern deep learning (e.g. some deep learning applications communicate 200MB for each worker \cite{1856.07238}). It is well-known \cite{1502.00251, 1602.04745, 1604.03077} that in many practical applications with common computing architectures, communication takes much more time than computation, creating a bottleneck of the entire training system.

**Communication Reduction.** Several solutions were suggested in the literature as a remedy to this problem. In one strain of work, the issue is addressed by giving each worker “more work” to do, which results in a better communication-to-computation ratio. For example, one may use mini-batching to construct more powerful gradient estimators \cite{1706.05065}, define local problems for each worker to be solved by a more advanced local solver \cite{1706.05065, 1803.04491, 1806.01157}, or reduce communication frequency (e.g. by communicating only once \cite{1501.03445, 1507.05740} or once every few iterations \cite{1604.03077}). An orthogonal approach to the above efforts aims to reduce the size of the communicated vectors instead \cite{1502.00251, 1602.04745, 1604.03077, 1702.07814, 1704.04765} using various lossy (and randomly randomized) compression mechanisms, commonly known in the literature as quantization techniques. In their most basic form, these schemes decrease the number of bits used to represent floating point numbers forming the communicated $d$-dimensional vectors \cite{1704.04765, 1704.04765, 1706.05065}, thus reducing the size of the communicated message by a constant factor. Another possibility is to apply randomized sparsification masks to the gradients \cite{1704.04765, 1602.04745, 1604.03077}, or to rely on coordinate/block descent updates-rules, which are sparse by design \cite{1706.05065, 1803.04491, 1806.01157}.

One of the most important considerations in the area of compression operators is the compression-variance trade-off \cite{1704.04765, 1704.04765, 1802.05958}. For instance, while random dithering approaches attain up to $O(d^{1/2})$ compression \cite{1502.00251, 1602.04745, 1604.03077}, the most aggressive schemes reach $O(d)$ compression by sending a constant number of bits per iteration only \cite{1704.04765, 1602.04745, 1604.03077}. However, the more compression is applied, the more information is lost, and the more will the quantized vector differ from the original vector we want to communicate, increasing its statistical variance. Higher variance implies slower convergence \cite{1502.00251, 1802.05958}, i.e., more communication rounds. So, ultimately, compression approaches offer a trade-off between the communication cost per iteration and the number of communication rounds. Remarkably, for carefully constructed compression strategies and appropriately designed training algorithms, this trade-off, while generally favouring some level of compression in practice, can be captured also theoretically \cite{1606.01213}.
2 Summary of Contributions

- **New Compression Operators.** We construct a new “natural compression” operator (see Sec 3 Def 1) based on a randomized rounding scheme in which each float of the compressed vector is rounded to a (positive or negative) power of 2. As a by-product, natural compression can get away with communicating the exponents and signs of the original floats only, which can be done with no computation effort beyond disposing off the mantissa and performing a bit-shift on the exponent. Importantly, natural compression enjoys a provably small variance, \(75/8\) (see Thm 1), which means that theoretical convergence results of SGD-type methods are essentially unaffected. At the same time, substantial savings are obtained in the amount of communicated bits per iteration (3.56x less for float32 and 5.82x less for float64). In addition, we utilize these insights and develop a new random dithering operator—natural dithering—which is exponentially better than standard random dithering (see Thm 3). Finally, our new compression techniques can be combined with existing compression and sparsification operators for a more dramatic combined effect (see Thm 3.3).

- **Computation-Free Simple Low-Level Implementation.** As we show in Sec 3.2 apart from a randomization procedure (which is inherent in all unbiased compression operators), natural compression is computation-free. Natural compression essentially amounts to the trimming of the mantissa even averaging) integers, new update compression methods are needed which can supply outputs in integer format. Our natural compression mechanism is the first that is provably able to operate in the SwitchML framework as it communicates integers only: the sign, plus the bits forming the exponent of a float. Moreover, having bounded (and small) variance, it is compatible with existing distributed training methods.

- **Proof-of-Concept System with In-Network Aggregation (INA).** The recently proposed SwitchML [40] system alleviates the communication bottleneck via in-network aggregation (INA) of gradients. However, since current programmable network switches are only capable of adding (not subtracting) integers, new update compression methods are needed which can supply outputs in an integer format. Our natural compression mechanism is the first that is provably able to operate in the SwitchML framework as it communicates integers only: the sign, plus the bits forming the exponent of a float. Moreover, having bounded (and small) variance, it is compatible with existing distributed training methods.

- **Theorem of generalized quantized SGD.** We provide a convergence theory for a distributed SGD method (see Algorithm 1), allowing for compression both at the worker and master side. Moreover, the compression operators compatible with our theory form a large family (operators \(C \in \mathcal{B} (\omega)\) for some finite \(\omega \geq 0\); see Definition 2). This enables safe experimentation with existing and facilitates the development of new compression operators fine-tuned to specific deep learning model architectures. Our convergence result (Thm 4) applies to smooth and non-convex functions, and our rates predict linear speed-up with respect to the number of machines.

- **Experiments.** We observe superior behavior compared to the state-of-the-art.

3 Natural Compression

We define a new (randomized) compression technique, which we call natural compression. This is fundamentally a function mapping \(t \in \mathbb{R}\) to a random variable \(C_{\text{nat}}(t) \in \mathbb{R}\). In case of vectors \(x = (x_1, \ldots, x_d) \in \mathbb{R}^d\) we apply it in an element-wise fashion: \((C_{\text{nat}}(x))_i = C_{\text{nat}}(x_i)\).

Natural compression \(C_{\text{nat}}\) performs a randomized rounding of its input \(t \in \mathbb{R}\) to one of the two closest integer powers of 2. Given nonzero \(t\), let \(\alpha \in \mathbb{R}\) be such that \(|t| = 2^\alpha\) (i.e., \(\alpha = \log_2 |t|\)). Then

\[
2^{\lceil \alpha \rceil} \leq |t| = 2^\alpha \leq 2^{\lceil \alpha \rceil}
\]

and we round \(t\) to either \(\text{sign}(t)2^{\lceil \alpha \rceil}\), or to \(\text{sign}(t)2^{\lceil \alpha \rceil}\). When \(t = 0\), we set \(C_{\text{nat}}(0) = 0\). The probabilities are chosen so that \(C_{\text{nat}}(t)\) is an unbiased estimator of \(t\), i.e., \(E[C_{\text{nat}}(t)] = t\) for all \(t\).

Example. For instance, \(t = -2.75\) will be rounded to either \(-4\) or \(-2\) (since \(-2^2 \leq -2.75 \leq -2^1\)), and \(t = 0.75\) will be rounded to either \(\frac{1}{2}\) or \(1\) (since \(-2^{-1} \leq 0.75 \leq 2^0\)). As a consequence, if \(t\) is an integer power of 2, then \(C_{\text{nat}}\) will leave \(t\) unchanged. See Fig 4 for a graphical illustration.

Definition 1 (Natural compression). Natural compression is a random function \(C_{\text{nat}} : \mathbb{R} \mapsto \mathbb{R}\) defined as follows. We set \(C_{\text{nat}}(0) = 0\). If \(t \neq 0\), we let

\[
C_{\text{nat}}(t) := \begin{cases} 
\text{sign}(t) \cdot 2^{\lceil \log_2 |t| \rceil}, & \text{with probability } p(t) := \frac{2^{\lceil \log_2 |t| \rceil} - |t|}{2^{\lceil \log_2 |t| \rceil}}, \\
\text{sign}(t) \cdot 2^{\lceil \log_2 |t| \rceil}, & \text{with probability } 1 - p(t),
\end{cases}
\]

3
We now explain that performing natural compression of a real number in a binary floating point
format is computationally cheap. In particular, excluding the randomization step, \( C_{\text{nat}} \) amounts to simply dispensing off the mantissa in the binary representation. The most common computer format for real numbers, binary32 (resp. binary64) of the IEEE 754 standard, represents each number with 32 (resp. 64) bits, where the first bit represents the sign, 8 (resp. 11) bits are used for the exponent, and the remaining 23 (resp. 52) bits are used for the mantissa. A scalar \( t \in \mathbb{R} \) is represented in the form \( (s,e_7,e_6,\ldots,e_0,m_1,m_2,\ldots,m_{23}) \), where \( s,e_i,m_j \in \{0,1\} \) are bits, via the relationship

\[
t = (-1)^s \times 2^{e-127} \times (1 + m), \quad e = \sum_{i=0}^{7} e_i 2^i, \quad m = \sum_{j=1}^{23} m_j 2^{-j},
\]

where \( s \) is the sign, \( e \) is the exponent and \( m \) is the mantissa.

Alternatively, \( C_{\text{nat}}(t) = \text{sign}(t) \cdot 2^{\lfloor \log_2 |t| \rfloor} (1 + \lambda(t)) \), where \( \lambda(t) \sim \text{Bernoulli}(1 - p(t)) \); that is, \( \lambda(t) = 1 \) with prob. \( 1 - p(t) \) and \( \lambda(t) = 0 \) with prob. \( p(t) \). The key properties of any (unbiased) compression operator are variance, ease of implementation, and compression level. We next characterize the remarkably low variance of \( C_{\text{nat}} \) in Sec 3.1 and describe an (almost) effortless and natural implementation, and the compression it offers in Sec 3.2.

3.1 Natural compression has a negligible variance: \( \omega = 1/\pi \)

We identify natural compression as belonging to a large class of unbiased compression operators with bounded second moment \([23][26][19]\), defined below.

**Definition 2** (Compression operators). A function \( C : \mathbb{R}^d \rightarrow \mathbb{R}^d \) mapping a deterministic input to a random vector is called a compression operator (on \( \mathbb{R}^d \)). We say that \( C \) is unbiased if

\[
E[C(x)] = x, \quad \forall x \in \mathbb{R}^d.
\]

If, moreover, there exists \( \omega \geq 0 \) such that

\[
E \|C(x)\|^2 \leq (\omega + 1) \|x\|^2, \quad \forall x \in \mathbb{R}^d,
\]

we say that \( C \) has bounded second moment. If \( C \) satisfies both \( (4) \) and \( (5) \), we will write \( C \in \mathbb{B}(\omega) \).

Note that \( \omega = 0 \) implies \( C(x) = x \) almost surely. It is easy to see \( (5) \) that the variance of \( C(x) \in \mathbb{B}(\omega) \) is bounded as: \( E \|C(x) - x\|^2 \leq \omega \|x\|^2 \). If this holds, we say that \( C \) has variance \( \omega \). The importance of \( \mathbb{B}(\omega) \) stems from two observations. First, operators from this class are known to be compatible with several optimization algorithms \([26][19]\). Second, this class includes many compression operators used in practice, including \([2][48][47][51]\). In general, the larger \( \omega \) is, the higher compression level might be achievable, and the worse impact compression has on the convergence speed.

The main result of this section says that the natural compression operator \( C_{\text{nat}} \) has variance \( 1/\pi \).

**Theorem 1.** \( C_{\text{nat}} \in \mathbb{B}(1/\pi) \).

Consider now a similar unbiased randomized rounding operator to \( C_{\text{nat}} \); but one that rounds to one of the nearest integers (as opposed to integer powers of 2). We call it \( C_{\text{int}} \). At first sight, this may seem like a reasonable alternative to \( C_{\text{nat}} \). However, as we show next, \( C_{\text{int}} \) does not have a finite second moment and is hence incompatible with existing optimization methods.

**Theorem 2.** There is no \( \omega \geq 0 \) such that \( C_{\text{int}} \in \mathbb{B}(\omega) \).

3.2 Natural compression: from 32 to 9 bits, with lightning speed

We now explain that performing natural compression of a real number in a binary floating point
format is computationally cheap.
Example 2. A binary32 representation of \( t = -2.75 \) is visualized in Fig. 2. In this case, \( s = 1, e_7 = 1, m_2 = m_3 = 1 \) and hence \( t = (-1)^s \times 2^{e_7-127} - (1 + m) = -1 \times 2 \times (1 + 2^{-2} + 2^{-3}) = -2.75 \). It is clear from (9) that \( 0 \leq m < 1 \), and hence \( 2^{-127} \leq |t| < 2^{-126} \) (compare this with (2)). Moreover, \( p(t) = \frac{2^{-126} - |t|}{2^{-127}} = 2 - |t|2^{127-e} = 1 - m \). Hence, natural compression of \( t \) represented as binary32 is given as follows:

\[
C_{\text{nat}}(t) = \begin{cases} (-1)^s \times 2^{e-127}, & \text{with probability } 1-m, \\ (-1)^s \times 2^{e-126}, & \text{with probability } m. 
\end{cases}
\]

Observe that \((-1)^s \times 2^{e-127}\) is obtained from \( t \) by setting the mantissa \( m \) to zero, and keeping both the sign \( s \) and exponent \( e \) unchanged. Similarly, \((-1)^s \times 2^{e-126}\) is obtained from \( t \) by setting the mantissa \( m \) to zero, keeping the sign \( s \), and increasing the exponent by one, which amounts to a simple shift of the bits forming the exponent to the left by one spot. Hence, both values can be computed from \( t \) essentially without any computation.

Communication savings. In summary, in case of binary32, the output \( C_{\text{nat}}(t) \) of natural compression is encoded using the 8 bits in the exponent and an extra bit for the sign. This is 3.56× less communication. In case of binary64, we only need 11 bits for the exponent and 1 bit for the sign, and this is 5.82× less communication.

3.3 Compatibility with other compression techniques

We start with a simple but useful observation about composition of compression operators.

Theorem 3. If \( C_1 \in \mathbb{B}(\omega_1) \) and \( C_2 \in \mathbb{B}(\omega_2) \), then \( C_1 \circ C_2 \in \mathbb{B}(\omega_{12}) \), where \( \omega_{12} = \omega_1 \omega_2 + \omega_1 + \omega_2 \), and \( C_1 \circ C_2 \) is the composition defined by \( (C_1 \circ C_2)_i(x) = C_1(C_2(x)) \).

Combining this result with Thm. 1, we observe that for any \( C \in \mathbb{B}(\omega) \), we have \( C_{\text{nat}} \circ C \in \mathbb{B}(\omega/s + 1/s) \). Since \( C_{\text{nat}} \) offers substantial communication savings with only a negligible effect on the variance of \( C \), a key use for natural compression beyond applying it as the sole compression strategy (e.g., for SwitchML [40]) is to deploy it with other effective techniques as a final compression mechanism (e.g., with the optimized sparsifiers [28, 47], or with [4, 44]), boosting the performance of the system even further. However, our technique will be useful also as a post-compression mechanism for compressions that do not belong to \( \mathbb{B}(\omega) \) (e.g., TopK sparsifier [44]). The same comments apply to the natural dithering operator \( D^{p,s}_{\text{nat}} \), defined in the next section.

4 Natural Dithering

Motivated by the natural compression introduced in Sec. 3 here we propose a new random dithering operator which we call natural dithering. However, it will be useful to introduce a more general dithering operator, one generalizing both the natural and the standard dithering operators. For \( 1 \leq p \leq +\infty \), let \( \|x\|_p \) be \( p \)-norm: \( \|x\|_p := (\sum_i |x_i|^p)^{1/p} \).

Definition 3 (General dithering). The general dithering operator with respect to the \( p \)-norm and with \( s \) levels \( 0 = l_s < l_{s-1} < l_{s-2} < \cdots < l_1 < l_0 = 1 \), denoted \( D^{p,s}_{\text{gen}} \), is defined as follows. Let \( x \in \mathbb{R}^d \). If \( x = 0 \), we let \( D^{p,s}_{\text{gen}}(x) = 0 \). If \( x \neq 0 \), we let \( y_i := x_i/\|x\|_p \) for all \( i \in [d] \). Assuming \( l_{u+1} \leq y_i \leq l_u \) for some \( u \in \{0, 1, \ldots, s - 1\} \), we let

\[
\left(D^{p,s}_{\text{gen}}(x)\right)_i = C(\|x\|_p) \times \text{sign}(x_i) \times \xi(y_i),
\]

(7)

where \( C \in \mathbb{B}(\omega) \) for some \( \omega \geq 0 \) and \( \xi(y_i) \) is a random variable equal to \( l_u \) with probability \( \frac{l_u - y_i}{l_u - l_{u+1}} \), and to \( l_{u+1} \) with probability \( \frac{y_i - l_{u+1}}{l_u - l_{u+1}} \). Note that \( \mathbb{E}[\xi(y_i)] = y_i \).
C∈C_{nat} with compression operators.

Table 1: The overall speedup of distributed SGD with compression on nodes via Theorem 4.

<table>
<thead>
<tr>
<th>Approach</th>
<th>( C_{\text{nat}} )</th>
<th>No. iterations</th>
<th>Bits per 1 iter.</th>
<th>Speedup Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>identity</td>
<td>( T'(\omega W) = O((\omega W + 1)^3) )</td>
<td>( 32d )</td>
<td>3.2 ( \times ) 3.6 ( \times )</td>
</tr>
<tr>
<td>New</td>
<td>( S^4 )</td>
<td>( \frac{2}{7} ) ( \omega ) ( \cdot ) ( d ) ( q )</td>
<td>( (33 + \log d)q )</td>
<td>0.6 ( \times ) 6.0 ( \times )</td>
</tr>
<tr>
<td>Sparsification</td>
<td>( C_{\text{nat}} \circ S^4 )</td>
<td>( \frac{2}{7} ) ( \omega ) ( \cdot ) ( d ) ( q )</td>
<td>( (10 + \log d)q )</td>
<td>1.0 ( \times ) 10.7 ( \times )</td>
</tr>
<tr>
<td>Dithering</td>
<td>( D_{\text{nat}}^{p,s} )</td>
<td>( (1 + \kappa d) / (2^{1-s}) ) ( \omega ) ( \cdot ) ( d ) ( q )</td>
<td>( 31 + d(2 + s) )</td>
<td>1.8 ( \times ) 15.9 ( \times )</td>
</tr>
</tbody>
</table>

Theorem 5. Fixing \( s \), natural dithering \( D_{\text{nat}}^{p,s} \) has \( O(2^{s-1}/s) \) times smaller variance than standard dithering \( D_{\text{sta}}^{p,u} \). Fixing \( \omega \), if \( u = 2^{s-1} \), then \( D_{\text{sta}}^{p,u} \in \mathbb{B}(\omega) \) implies that \( D_{\text{nat}}^{p,s} \in \mathbb{B}(\omega + 1) - 1 \).

5 Distributed SGD with Bidirectional Compression

There are several stochastic gradient-based methods [38, 6, 12, 31] for solving (1) that are compatible with compression operators \( C \in \mathbb{B}(\omega) \), and hence also with our natural compression \( (C_{\text{nat}}) \) and natural...
We assume repeated access to unbiased stochastic gradients $g_k$. Distributed SGD with bi-directional compression is given by

$$C_w = \begin{cases} \frac{1}{2} & \text{if no compression is used} \\ \frac{1}{M} & \text{if compression is used by the workers} \\ \frac{1}{2M} & \text{if compression is used by both sides} \end{cases}$$

where $C_w \in B(\omega_M)$ is the compression operator used by the master node, $C_{W_i} \in B(\omega_{W_i})$ are the compression operators used by the workers and $\omega_W := \max_{i \in [n]} \omega_{W_i}$. The main theorem follows:

**Theorem 6.** Let $C_M \in B(\omega_M), C_{W_i} \in B(\omega_{W_i})$ and let the stepsizes be set to $\eta^k = \eta \in (0, 2/L)$, where $\alpha, \beta$ are as in (8). If $a$ is picked uniformly at random from $\{0, 1, \ldots, T - 1\}$, then

$$E \left[ \frac{1}{\| f(x^a) \|} \right] \leq \frac{2(f(x^a) - f(x^*))}{\eta(2 - 2\eta)T} + \frac{\alpha L_n}{2\beta L_n},$$

where $x^*$ is an opt. solution of (1). In particular, if we fix any $\varepsilon > 0$ and choose $\eta = \frac{\varepsilon}{L(\alpha + \varepsilon \beta)}$ and $T \geq \frac{2L(f(x^a) - f(x^*))}{\varepsilon^2(\alpha + \varepsilon \beta)}$, then $E \left[ \frac{1}{\| f(x^a) \|} \right] \leq \varepsilon$.

The above theorem has some interesting consequences. First, notice that posits a $O(1/T)$ convergence of the gradient norm to the value $\frac{\alpha L_n}{2\beta L_n}$, which depends linearly on $\alpha$. In view of (8), the more compression we perform, the larger this value. More interestingly, assume now that the same compression operator is used at each worker: $C_M = C_{W_i}$. Let $C_W \in B(\omega_W)$ and $C_M \in B(\omega_M)$ be the compression on master side. Then, $T(\omega_M, \omega_W) := 2L(f(x^a) - f(x^*))\varepsilon^{-2}(\alpha + \varepsilon \beta)$ is its iteration complexity. In the special case of equal data on all nodes, i.e., $\omega_{W_i} = 0$, we get $\alpha = (\omega_M + 1)\sigma^2/n$ and $\beta = (\omega_M + 1)(1 + \omega_W/n)$. If no compression is used, then $\omega_W = \omega_M = 0$ and $\alpha + \varepsilon \beta = \sigma^2/n + \varepsilon$. So, the relative slowdown of Algorithm 1 used with compression compared to Algorithm 1 used without compression is given by

$$T(\omega_M, \omega_W)/T(0, 0) = \frac{\left(\frac{(\omega_M + 1)\sigma^2/n + (1 + \omega_W/n)\varepsilon}{\omega_M + 1} \in (\omega_M + 1, (\omega_M + 1)(\omega_W + 1))}\right.}.$$
6 System Evaluation

To verify the theoretical properties of our approach in practice, we built a proof-of-concept system and provide evaluation results. In particular, we focus on illustrating convergence behavior, training throughput improvement, and transmitted data reduction.

Experimental setup. Our implementation builds upon the concept of In-Network Aggregation \cite{40}. Appendix A describes the implementation details. We run the workers on 8 machines configured with 1 NVIDIA P100 GPU, dual CPU Intel Xeon E5-2630 v4 at 2.20GHz, and 128 GB of RAM. The machines run Ubuntu (Linux kernel 4.4.0-122) and CUDA 9.0. Following \cite{40}, we balance the workers with 8 aggregators running on machines configured with dual Intel Xeon Silver 4108 CPU at 1.80 GHz. Each machine uses a 10 GbE network interface and has CPU frequency scaling disabled. The chunks of compressed gradients sent by workers are uniformly distributed across all aggregators. This setup ensures that workers can fully utilize their network bandwidth and match the performance of a programmable switch. We leave the switch-based implementation for future work.

Our experiments execute the standard CNN benchmark \cite{46}. We summarize the hyperparameters setting in Appendix B.1.1. We further present results for two more variations of our implementation: one without compression (providing the baseline for In-Network Aggregation), and the other with deterministic rounding to the nearest power of 2.

Results. We first illustrate the convergence behavior by training ResNet110 and AlexNet models on CIFAR10 dataset. Fig 4 shows the test accuracy over time. We note that natural compression lowers training time by $\sim 26\%$ and $66\%$, resp., compared to using no compression, while the accuracy matches the results in \cite{18} with the same hyperparameters setting. Moreover, compression using deterministic rounding (not shown) instead of stochastic rounding does not further reduce training time. In addition, our approaches do not affect the convergence speed in terms of training loss as predicted by theory, even when we use $16 \times$ fewer levels for $D_{\text{nat}}^{p,s}$ w.r.t. $D_{\text{sta}}^{p,s}$; see Appendix B.3.

Next, we report the speedup measured in average training throughput while training benchmark CNN models on Imagenet dataset for one epoch. The throughput is calculated as the total number of images processed divided by the time elapsed. Fig 5 shows the speedup normalized by the training throughput of the baseline, that is, TensorFlow + Horovod using the NCCL communication library. We further break down the speedup by showing the relative speedup of In-Network Aggregation, which performs no compression but reduces the volume of data transferred (shown below). We also show the effects of deterministic rounding on throughput. Because deterministic rounding does not compute random
numbers, it provides some additional speedups. However, it may affect convergence. These results represent potential speedups in case the overheads of randomization were low, for instance, when using simple lookup for pre-computed randomness. We observe that the communication-intensive models (VGG, AlexNet) benefit more from quantization as compared to the computation-intensive models (GoogleNet, Inception, ResNet). These observations are consistent with prior work [2].

To quantify the data reduction benefits of natural compression, we measure the total volume of data transferred during training. Fig [6] shows that data transferred grows linearly over time, as expected. Natural compression saves 84% of data, which greatly reduces communication time.

Further details and additional experiments are presented in Appendix B.

References


[41] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. pages 1058–1062. ISCA, 2014.


Natural Compression for Distributed Deep Learning (Appendix)

For easy navigation through the Appendix, we provide a table of contents.

Contents

1 Introduction .................................................. 1

2 Summary of Contributions ............................... 3

3 Natural Compression ........................................ 3
   3.1 Natural compression has a negligible variance: \( \omega = 1/8 \) ........................................ 4
   3.2 Natural compression: from 32 to 9 bits, with lightning speed ........................................ 4
   3.3 Compatibility with other compression techniques .................................................. 5

4 Natural Dithering ............................................. 5

5 Distributed SGD with Bidirectional Compression ....... 6

6 System Evaluation ............................................ 8

A Implementation Details .................................... 14

B Extra Experiments ........................................... 15
   B.1 Convergence Tests on CIFAR 10 ......................... 15
      B.1.1 AlexNet Hyperparameters: ......................... 15
      B.1.2 ResNet Hyperparameters: ......................... 15
      B.1.3 DenseNet Hyperparameters: ....................... 15
   B.2 \( D_{\text{nat}}^{p,s} \) vs. \( D_{\text{sta}}^{p,u} \): Empirical Variance ........................................ 16
      B.2.1 \( D_{\text{nat}}^{p,s} \) has exponentially better variance ........................................ 16
      B.2.2 \( D_{\text{nat}}^{p,s} \) needs exponentially less levels to achieve the same variance .................... 16
      B.2.3 \( D_{\text{sta}}^{p,s} \) can outperform \( D_{\text{nat}}^{p,s} \) in the big \( s \) regime ........................................ 16
      B.2.4 Compressing gradients ............................... 17
   B.3 Different Compression Operators ....................... 18

C Details and Proofs for Sections 3 and 4 ............... 20
   C.1 Proof of Theorem 1 .................................... 20
   C.2 Proof of Theorem 2 .................................... 20
   C.3 Proof of Theorem 3 .................................... 21
   C.4 Proof of Theorem 4 .................................... 21
   C.5 Proof of Theorem 5 .................................... 21
   C.6 Natural Compression and Dithering Allow for Fast Aggregation ............................ 22
D Details and Proofs for Section 5
D.1 Assumptions and Definitions ........................................... 23
D.2 Description of Algorithm ............................................ 23
D.3 Three Lemmas Needed for the Proof of Theorem 6 ........ 23
D.4 Proof of Theorem .................................................... 25
D.5 A Different Stepsize Rule for Theorem 6 ................ 26
D.6 SGD with Bidirectional Compression: Four Models .......... 26
   D.6.1 Model 1 ...................................................... 27
   D.6.2 Model 2 ...................................................... 27
   D.6.3 Model 3 ...................................................... 28
   D.6.4 Model 4 ...................................................... 28
   D.6.5 Communication strategies used in Tables 1356 ........ 28
D.7 Sparsification - Formal Definition ................................. 28

E Limitations and Extensions ............................................. 29
A Implementation Details

We implement the natural compression operator within the Gloo communication library [10], as a drop-in replacement for the ring all-reduce routine. Our implementation is in C++. We integrate our communication library with Horovod and, in turn, with TensorFlow. We follow the same communication strategy introduced in SwitchML [40], which aggregates the deep learning model’s gradients using In-Network Aggregation on programmable network switches. We choose this strategy because natural compression is a good fit for the capabilities of this class of modern hardware, which only supports basic integer arithmetic, simple logical operations and limited storage.

A worker applies the natural compression operator to quantize gradient values and sends them to the aggregator component. As in SwitchML, an aggregator is capable of aggregating a fixed-length array of gradient values at a time. Thus, the worker sends a stream of network packets, each carrying a chunk of compressed values. For a given chunk, the aggregator awaits all values from every worker; then, it restores the compressed values as integers, aggregates them and applies compression to quantize the aggregated values. Finally, the aggregator multicasts back to the workers a packet of aggregated values.

For implementation expedience, we prototype the In-Network Aggregation as a server-based program implemented atop DPDK [8] for fast I/O performance. We leave to future work a complete P4 implementation for programmable switches; however, we note that all operations (bit shifting, masking, and random bits generation) needed for our compression operator are available on programmable switches.

Difference in Implementation. We carefully optimize our implementation using modern x86 vector instructions (AVX512) to minimize the overheads in doing compression. To fit the byte length and access memory more efficiently, we compress a 32-bit floating point number to an 8-bit representation, where 1 bit is for the sign and 7 bits are for the exponent. The aggregator uses 64-bit integers to store the intermediate results, and we choose to clip the exponents in the range of $-50 \sim 10$. As a result, we only use 6 bits for exponents. The remaining one bit is used to represent zeros. Note that it is possible to implement 128-bit integers using two 64-bit integers, but we found that, in practice, the exponent values never exceed the range of $-50 \sim 10$ (Figure 7).

Despite the optimization effort, we identify non-negligible 10 $\sim$ 15% overheads in doing random number generation used in stochastic rounding, which was also reported in [22]. We include the experimental results of our compression operator without stochastic rounding as a reference. There could be more efficient ways to deal with stochastic rounding, but we observe that doing deterministic rounding gives nearly the same training curve in practice.

![Figure 7: Histogram of exponents of gradients exchanged during the entire training process for ResNet110 (left) and Alexnet (right). Red lines denote the minimum and maximum exponent values of all gradients.](image-url)
B Extra Experiments

B.1 Convergence Tests on CIFAR 10

In order to validate that $C_{nat}$ does not incur any loss in performance, we trained various DNNs on the Tensorflow CNN Benchmark [46] on the CIFAR 10 dataset with and without $C_{nat}$ for the same number of epochs, and compared the test set accuracy. As mentioned earlier, the baseline for comparison is the default NCCL setting. We didn’t tune the hyperparameters. In all of the experiments, we used Batch Normalization, but no Dropout was used.

![Graph](image1)

(a) AlexNet (Batch size: 256, 512 and 1024)

![Graph](image2)

(b) ResNet (#layers: 20, 44 and 56)

![Graph](image3)

(c) DenseNet40 (k=12)

Figure 8: Convergence experiments on CIFAR10. $C_{nat}$ achieves significant speedup without incurring any accuracy loss. As expected, the communication intensive AlexNet (62.5 M parameters) benefits more from the compression than the computation intensive ResNets (< 1.7 M parameters) and DenseNet40 (1 M parameters). One should note that $C_{nat}$ is faster that QSGD [2] on these DNNs.

B.1.1 AlexNet Hyperparameters:

For AlexNet, we chose the optimizer as SGD with momentum with a momentum of 0.9. We trained on three minibatch sizes: 256, 512 and 1024 for 200 epochs. The learning rate was initially set to be 0.001, which was decreased by a factor of 10 after every 30 epoch.

B.1.2 ResNet Hyperparameters:

All the ResNets followed the training procedure as described in [18]. We used a weight decay of $10^{-4}$ and the optimizer was chosen to be SGD with momentum, with a momentum of 0.9. The minibatch size was fixed to be 128 for ResNet 20, and 256 for all the others. We train for a total of 64K iterations. We start with an initial learning rate of 0.1, and multiply it by 0.1 at 32K and 48K iterations.

B.1.3 DenseNet Hyperparameters:

We trained DenseNet40 ($k = 12$) and followed the same training procedure as described in the original paper [21]. We used a weight decay of $10^{-4}$ and the optimizer as SGD with momentum,
with a momentum of 0.9. We trained for a total of 300 epochs. The initial learning rate was 0.1, which was decreased by a factor of 10 at 150 and 225 epoch.

\subsection{B.2 $\mathcal{D}_{\text{nat}}^{p,s}$ vs. $\mathcal{D}_{\text{sta}}^{p,u}$: Empirical Variance}

In this section, we perform experiments to confirm that $\mathcal{D}_{\text{nat}}^{p,s}$ level selection brings not just theoretical but also practical performance speedup in comparison to $\mathcal{D}_{\text{sta}}^{p,u}$. We measure the empirical variance of $\mathcal{D}_{\text{sta}}^{p,u}$ and $\mathcal{D}_{\text{nat}}^{p,s}$. For $\mathcal{D}_{\text{nat}}^{p,s}$, we do not compress the norm, so we can compare just variance introduced by level selection. Our experimental setup is the following. We first generate a random vector $x$ of size $d = 10^5$, with independent entries with Gaussian distribution of zero mean and unit variance (we tried other distributions, the results were similar, thus we report just this one) and then we measure normalized empirical variance

$$\omega(x) := \frac{\|C(x) - x\|^2}{\|x\|^2}.$$

We provide boxplots, each for 100 randomly generated vectors $x$ using the above procedure. We perform this for $p = 1, p = 2$ and $p = \infty$. We report our findings in Fig 9, Fig 10 and Fig 11. These experimental results support our theoretical findings.

\subsubsection{B.2.1 $\mathcal{D}_{\text{nat}}^{p,s}$ has exponentially better variance}

In Fig 9, we compare $\mathcal{D}_{\text{nat}}^{p,s}$ and $\mathcal{D}_{\text{sta}}^{p,u}$ for $u = s$, i.e., we use the same number of levels for both compression strategies. In each of the three plots we generated vectors $x$ with a different norm. We find that natural dithering has dramatically smaller variance, as predicted by Theorem 5.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig9}
\caption{$\mathcal{D}_{\text{nat}}^{p,s}$ vs. $\mathcal{D}_{\text{sta}}^{p,u}$ with $u = s$.}
\end{figure}

\subsubsection{B.2.2 $\mathcal{D}_{\text{nat}}^{p,s}$ needs exponentially less levels to achieve the same variance}

In Fig 10, we set the number of levels for $\mathcal{D}_{\text{sta}}^{p,u}$ to $u = 2^{s-1}$. That is, we give standard dithering an exponential advantage in terms of the number of levels (which also means that it will need more bits for communication). We now study the effect of this change on the variance. We observe that the empirical variance is essentially the same for both, as predicted by Theorem 5.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig10}
\caption{$\mathcal{D}_{\text{nat}}^{p,s}$ vs. $\mathcal{D}_{\text{sta}}^{p,u}$ with $u = 2^{s-1}$.}
\end{figure}

\subsubsection{B.2.3 $\mathcal{D}_{\text{sta}}^{p,s}$ can outperform $\mathcal{D}_{\text{nat}}^{p,s}$ in the big $s$ regime}

We now remark on the situation when the number of levels $s$ is chosen to be very large (see Fig 11). While this is not a practical setting as it does not provide sufficient compression, it will serve as an illustration of a fundamental theoretical difference between $\mathcal{D}_{\text{sta}}^{p,s}$ and $\mathcal{D}_{\text{nat}}^{p,s}$ in the $s \to \infty$ limit.
which we want to highlight. Note that while \( D_{sta}^{p,s} \) converges to the identity operator as \( s \to \infty \), which enjoys zero variance, \( D_{nat}^{p,s} \) converges to \( C_{nat} \) instead, with variance that can’t reduce below \( \omega = \frac{1}{8} \). Hence, for large enough \( s \), one would expect, based on our theory, the variance of \( D_{nat}^{p,s} \) to be around \( \frac{1}{s} \), while the variance of \( D_{sta}^{p,s} \) to be closer to zero. In particular, this means that \( D_{sta}^{p,s} \) can, in a practically meaningless regime, outperform \( D_{nat}^{p,s} \). In Fig 11 we choose \( p = \infty \) and \( s = 32 \) (this is large). Note that, as expected, the empirical variance of both compression techniques is small, and that, indeed, \( D_{sta}^{p,s} \) outperforms \( D_{nat}^{p,s} \).

**Figure 11:** When \( p = \infty \) and \( s \) is very large, the empirical variance of \( D_{sta}^{p,s} \) can be smaller than that of \( D_{nat}^{p,s} \). However, in this case, the variance of \( D_{nat}^{p,s} \) is already negligible.

### B.2.4 Compressing gradients

We also performed identical to those reported above, but with a different generation technique of the vectors \( x \). In particular, instead of a synthetic Gaussian generation, we used gradients generated by our optimization procedure as applied to the problem of training several deep learning models. Our results were essentially the same as the ones reported above, and hence we do not include them.
B.3 Different Compression Operators

We report additional experiments where we compare our compression operator to previously proposed ones. These results are based on a Python implementation of our methods running in PyTorch as this enabled a rapid direct comparisons against the prior methods. We compare against no compression, random sparsification, and random dithering methods. We compare on MNIST and CIFAR10 datasets. For MNIST, we use a two-layer fully connected neural network with RELU activation function. For CIFAR10, we use VGG11 with one fully connected layer as the classifier. We run these experiments with 4 workers and batch size 32 for MNIST and 64 for CIFAR10. The results are averages over 3 runs.

We tune the step size for SGD for a given “non-natural” compression. Then we use the same step size for the “natural” method. Step sizes and parameters are listed alongside the results.

Figures 12 and 13 illustrate the results. Each row contains four plots that illustrate, left to right, (1) the test accuracy vs. the volume of data transmitted (measured in bits), (2) the test accuracy over training epochs, (3) the training loss vs. the volume of data transmitted, and (4) the training loss over training epochs.

One can see that in terms of epochs, we obtain almost the same result in terms of training loss and test accuracy, sometimes even better. On the other hand, our approach has a huge impact on the number of bits transmitted from workers to master, which is the main speedup factor together with the speedup in aggregation if we use In-Network Aggregation (INA). Moreover, with INA we compress updates also from master to nodes, hence we send also fewer bits. These factors together bring significant speedup improvements, as illustrated in Fig 5.

(a) No Additional compression, step size 0.1.

(b) Random sparsification, step size 0.04, sparsity 10%.

(c) Random sparsification with non-uniform probabilities \( \epsilon = 0.7 \), step size 0.04, sparsity 10%.

(d) Random dithering, step size 0.08, \( s = 8 \), \( u = 2^7 \), second norm.

**Figure 12:** CIFAR10 with VGG11.
(a) No Additional compression, step size 0.1.

(b) Random sparsification, step size 0.04, sparsity 10%.

(c) Random sparsification with non-uniform probabilities \[47\], step size 0.04, sparsity 10%.

(d) Random dithering, step size 0.01, \(s = 8, u = 2^7\), second norm.

**Figure 13:** MNIST with 2 fully connected layers.
C Details and Proofs for Sections 3 and 4

C.1 Proof of Theorem 1

By linearity of expectation, the unbiasedness condition (4) and the second moment condition (5) have the form

$$E[(C(x))_i] = x_i, \quad \forall x \in \mathbb{R}^d, \quad \forall i \in [d]$$

(11)

and

$$\sum_{i=1}^{d} E[(C(x))^2_i] \leq (\omega + 1) \sum_{i=1}^{d} x_i^2, \quad \forall x \in \mathbb{R}^d.$$  

(12)

Recall that $C_{\text{nat}}(t)$ can be written in the form

$$C_{\text{nat}}(t) = \text{sign}(t) \cdot 2^{\lfloor \log_2 |t| \rfloor} (1 + \lambda(t)).$$

(13)

where the last step follows since $p(t) = 2^{\lfloor \log_2 |t| \rfloor} - |t|$. Hence,

$$E[C_{\text{nat}}(t)] = E \left[ \text{sign}(t) \cdot 2^{\lfloor \log_2 |t| \rfloor} (1 + \lambda(t)) \right] = \text{sign}(t) \cdot 2^{\lfloor \log_2 |t| \rfloor} (1 + E[\lambda(t)]) = \text{sign}(t) \cdot 2^{\lfloor \log_2 |t| \rfloor} (1 + 1 - p(t)) = t,$$

which establishes unbiasedness (11).

In order to establish (12), it suffices to show that $E \left[ (C_{\text{nat}}(x))^2_i \right] \leq (\omega + 1)x_i^2$ for all $x_i \in \mathbb{R}$. Since by definition $(C_{\text{nat}}(x))_i = C_{\text{nat}}(x_i)$ for all $i \in [d]$, it suffices to show that

$$E \left[ (C_{\text{nat}}(t))^2 \right] \leq (\omega + 1)t^2, \quad \forall t \in \mathbb{R}.$$  

(14)

If $t = 0$ or $t = \text{sign}(t)2^{\alpha}$ with $\alpha$ being an integer, then $C_{\text{nat}}(t) = t$, and (14) holds as an identity with $\omega = 0$, and hence inequality (14) holds for $\omega = 1/8$. Otherwise $t = \text{sign}(t)2^{\alpha}$ where $\alpha := \lfloor \alpha \rfloor < \alpha < \lceil \alpha \rceil = a + 1$. With this notation, we can write

$$E \left[ (C_{\text{nat}}(t))^2 \right] = 2^a \frac{2^a + 1}{2^a} \frac{|t|}{t} + 2^{(a+1)} \frac{|t|^2 - 2^a}{2^a} = 2^a (3|t| - 2^{a+1}).$$

So,

$$\frac{E \left[ (C_{\text{nat}}(t))^2 \right]}{t^2} = \frac{2^a (3|t| - 2^{a+1})}{t^2} \leq \frac{2^a (3|t| - 2^{a+1})}{t^2} = \sup_{0 < t < 2^{a+1}} \frac{2^a (3|t| - 2^{a+1})}{t^2} \leq \sup_{1 < \theta < 2} \frac{3\theta - 2}{\theta^2}. $$

The optimal solution of the last maximization problem is $\theta = \frac{4}{3}$, with optimal objective value $\frac{9}{8}$. This implies that (14) holds with $\omega = \frac{1}{8}$.

C.2 Proof of Theorem 2

Let assume that there exists some $\omega < \infty$ for which $C_{\text{int}}$ is the $\omega$ quantization. Unbiased rounding to the nearest integer can be defined in the following way

$$C_{\text{int}}(x_i) := \begin{cases} \lfloor x_i \rfloor, \quad \text{with probability } p(x_i), \\ \lceil x_i \rceil, \quad \text{with probability } 1 - p(x_i), \end{cases}$$

where $p(x_i) = \lfloor x_i \rfloor - x_i$. Let’s take 1-D example, where $x \in (0, 1)$, then

$$E \left[ C_{\text{int}}(x^2) \right] = (1 - x)0^2 + x1^2 = x \leq \omega x^2,$$

which implies $\omega \geq 1/x$, thus taking $x \to 0^+$, one obtains $\omega \to \infty$, which contradicts the existence of finite $\omega$. 

20
C.3 Proof of Theorem 3

The main building block of the proof is the tower property of mathematical expectation. The tower property says: If $X$ and $Y$ are random variables, then $E[X] = E[E[X | Y]]$. Applying it to the composite compression operator $C_1 \circ C_2$, we get

$$E[(C_1 \circ C_2)(x)] = E[E[C_1(C_2(x)) \mid C_2(x)]] E[C_2(x)] \overset{[5]}{=} x.$$  

For the second moment, we have

$$E\left[\left\| (C_1 \circ C_2)(x) \right\|^2 \right] = E\left[ E\left[ \left\| C_1(C_2(x)) \right\|^2 \mid C_2(x) \right] \right] \overset{[5]}{\leq} (\omega_2 + 1) E\left[ \left\| C_1(x) \right\|^2 \right] \overset{[5]}{\leq} (\omega_1 + 1)(\omega_2 + 1) \left\| x \right\|^2,$$

which concludes the proof.

C.4 Proof of Theorem 4

Unbiasedness of $D_{\text{nat}}^{p,s}$ is a direct consequence of unbiasedness of $D_{\text{gen}}^{C,p,s}$.

For the second part, we first establish a bound on the second moment of $\xi$:

$$E\left[ \xi \left( \frac{x_i}{\|x\|_p} \right)^2 \right] \leq 1 \left( \frac{|x_i|}{\|x\|_p} \geq 2^{1-s} \right) \frac{9}{8} \frac{|x_i|^2}{\|x\|_p^2} + 1 \left( \frac{|x_i|}{\|x\|_p} < 2^{1-s} \right) \frac{|x_i|^2}{\|x\|_p} 2^{1-s} \frac{9}{8} \left( \frac{|x_i|^2}{\|x\|_p^2} + \sum_{i=1}^{d} 1 \left( \frac{|x_i|}{\|x\|_p} < 2^{1-s} \right) \frac{|x_i|^2}{\|x\|_p} 2^{1-s} \right) \frac{9}{8} \left( \frac{|x|^2}{\|x\|_p^2} + \min \left\{ d^{1/2} 2^{1-s} \|x\|_p \|x\|, 2^{2-2s} d \right\} \right) \frac{9}{8} \left( \frac{|x|^2}{\|x\|_p^2} + \min \left\{ d^{1/2} 2^{1-s} \|x\|_p \|x\|, 2^{2-2s} d \|x\|_p^2 \right\} \right) \frac{9}{8} \left( \frac{9}{8} + d^{1/\min(p,2)} 2^{1-s} \min \left\{ 1, d^{1/\min(p,2)} 2^{1-s} \right\} \right) \|x\|^2,$$

where the last two inequalities follow from the Hölder’s inequality $\|x\|_p \leq d^{1/p-1/2} \|x\|$ for $0 \leq p \leq 1$ and from the fact that $\|x\|_p \leq \|x\|$ for $p \geq 2$.

C.5 Proof of Theorem 5

The main building block of the proof is useful connection between $D_{\text{nat}}^{p,s}$ and $D_{\text{sta}}^{p,2^{s-1}}$, which can be formally written as

$$D_{\text{nat}}^{p,s}(x) \overset{D}{=} C_{\text{nat}}(\|x\|_p) \cdot \text{sign}(x) \cdot C_{\text{nat}}(\xi(x)),$$

where $(\xi(x))_i = \xi(x_i/\|x\|_p)$ with levels $0, 1/2^{s-1}, 2/2^{s-1}, \ldots, 1$. The reason why these two random variables have the same distribution is that they both use $C_{\text{nat}}$ for norm. Moreover, $C_{\text{nat}}(\xi(x_i/\|x\|_p))$. 21
Figure 14: 1D visualization of the workings of natural dithering $D_{\text{nat}}^{p,s}$ and standard dithering $D_{\text{sta}}^{p,u}$ with $u = 2^{s-1}$, with $s = 4$. Notice that the numbers standard dithering rounds to, i.e., $0, 1/8, 2/8, \ldots, 7/8, 1$, form a superset of the numbers natural dithering rounds to, i.e., $0, 2^{-3}, 2^{-2}, 2^{-1}, 1$. Importantly, while standard dithering uses $u = 2^{4-1} = 8$ levels (i.e., intervals) to achieve a certain fixed variance, natural dithering only needs $s = 4$ levels to achieve the same variance. This is an exponential improvement in compression (see Theorem 5 for the formal statement).

where $\xi(x_i/\|x\|_p)$ has levels $0, 1/2^{s-1}, 2/2^{s-1}, \ldots, 1$ has the exact same distribution as $\xi(x_i/\|x\|_p)$ with levels $0, 1/2^{s-1}, 1/2^{s-2}, \ldots, 1$, as they produce the same levels and composition of two unbiased random variables is unbiased due to tower property. Graphical visualization can be found in Fig 14.

Equipped with this, we can proceed with

$$E \left[ \left\| \xi(x_i/\|x\|_p) \right\|^2 \right] \leq E \left[ \left| C_{\text{nat}}(\|x\|_p) \cdot \text{sign}(x) \cdot C_{\text{nat}}(\xi(x)) \right|^2 \right]$$

$$= E \left[ C_{\text{nat}}(\|x\|_p)^2 \cdot E \left[ \left| C_{\text{nat}}(\xi(x)) \right|^2 \right] \right]$$

$$\leq \frac{9^2}{8^2} E \left[ \left\| x_i/\|x\|_p \cdot \text{sign}(x) \xi(x) \right\|^2 \right]$$

$$= \frac{9^2}{8^2} E \left[ \left\| D_{\text{sta}}^{p,2^{s-1}} \right\|^2 \right]$$

$$\leq \frac{9^2}{8^2}(\omega + 1),$$

which concludes the proof.

C.6 Natural Compression and Dithering Allow for Fast Aggregation

Besides communication savings, our new compression operators $C_{\text{nat}}$ (natural compression) and $D_{\text{nat}}^{p,s}$ (natural dithering) bring another advantage, which is ease of aggregation. Firstly, our updates allow in-network aggregation on a primitive switch, which can speed up training by up to 300% itself. Moreover, our updates are so simple that if one uses integer format on the master side for update aggregation, then our updates have just one non-zero bit, which leads to additional speed up. For this reason, one needs to operate with at least 64 bits during the aggregation step, which is the reason why we also do $C_{\text{nat}}$ compression on the master side; and hence we need to transmit just exponent to workers. Moreover, the translation from floats to integers and back is computation-free due to structure of our updates. Lastly, for $D_{\text{nat}}^{p,s}$ compression we obtain additional speed up with respect to standard randomized dithering $D_{\text{sta}}^{p,s}$ as our levels are computationally less expensive due to their natural compatibility with floating points. In addition, for effective communication one needs to communicate signs, norm and levels as a tuple for both $D_{\text{nat}}^{p,s}$ and $D_{\text{sta}}^{p,s}$, which needs to be then multiplied back on the master side. For $D_{\text{nat}}^{p,s}$, this is just the summation of exponents rather than actual multiplication as is the case for $D_{\text{sta}}^{p,s}$.
D Details and Proofs for Section 5

D.1 Assumptions and Definitions

Formal definitions of some concepts used in Section follows:

**Definition 4.** Let \( f_i : \mathbb{R}^d \rightarrow \mathbb{R} \) be fixed function. A stochastic gradient for \( f_i \) is a random function \( g_i(x) \) so that \( \mathbb{E}[g_i(x)] = \nabla f_i(x) \).

In order to obtain the rate, we introduce additional assumptions on \( g_i(x) \) and \( \nabla f_i(x) \).

**Assumption 1** (Bounded Variance). We say the stochastic gradient has variance at most \( \sigma_i^2 \) if \( \mathbb{E} \left[ \| g_i(x) - \nabla f_i(x) \|^2 \right] \leq \sigma_i^2 \) for all \( x \in \mathbb{R}^d \). Moreover, let \( \sigma^2 = \frac{1}{n} \sum_{i=1}^{n} \sigma_i^2 \).

**Assumption 2** (Similarity). We say the variance of gradient among nodes is at most \( \zeta_i^2 \) if \( \| \nabla f_i(x) - \nabla f(x) \|^2 \leq \zeta_i^2 \) for all \( x \in \mathbb{R}^d \). Moreover, let \( \zeta^2 = \frac{1}{n} \sum_{i=1}^{n} \zeta_i^2 \).

D.2 Description of Algorithm 1

Let us describe Algorithm 1. First, each worker computes its own stochastic gradient \( g_i(x^k) \), this is then compressed using a compression operator \( \mathcal{C}_{W_i} \) (this can be different for every node, for simplicity, one can assume that they are all the same) and send to the master node. The master node then aggregates the updates from all the workers, compress with its own operator \( \mathcal{C}_M \) and broadcasts update back to the workers, which update their local copy of the solution parameter \( x \).

Note that the communication of the updates can be also done in all-to-all fashion, which implicitly results in \( \mathcal{C}_M \) being the identity operator. Another application, which is one of the key motivations of our natural compression and natural dithering operators, is in-network aggregation [40]. In this setup, the master node is a network switch. However, current network switches can only perform addition (not even average) of integers.

D.3 Three Lemmas Needed for the Proof of Theorem 6

Before we proceed with the theoretical guarantees for Algorithm 1 in smooth non-convex setting, we first state three lemmas which are used to bound the variance of \( g^k \) as a stochastic estimator of the true gradient \( \nabla f(x^k) \). In this sense compression at the master-node has the effect of injecting additional variance into the gradient estimator. Unlike in SGD, where stochasticity is used to speed up computation, here we use it to reduce communication.

**Lemma 7** (Tower property + Compression). If \( \mathcal{C} \in \mathbb{B}(\omega) \) and \( z \) is a random vector, then

\[
\mathbb{E} \left[ \| \mathcal{C}(z) - z \|^2 \right] \leq \omega \mathbb{E} \left[ \| z \|^2 \right] ; \quad \mathbb{E} \left[ \| \mathcal{C}(z) \|^2 \right] \leq (\omega + 1) \mathbb{E} \left[ \| z \|^2 \right] .
\]

**Proof.** Recall from the discussion following Definition 2 that the variance of a compression operator \( \mathcal{C} \in \mathbb{B}(\omega) \) can be bounded as

\[
\mathbb{E} \left[ \| \mathcal{C}(x) - x \|^2 \right] \leq \omega \| x \|^2 , \quad \forall x \in \mathbb{R}^d .
\]

Using this with \( z = x \), this can be written in the form

\[
\mathbb{E} \left[ \| \mathcal{C}(z) - z \|^2 | z \right] \leq \omega \| z \|^2 , \quad \forall x \in \mathbb{R}^d ,
\]
which we can use in our argument:

$$E \left[ \|C(z) - z\|^2 \right] = E \left[ E \left[ \|C(z) - z\|^2 \mid z \right] \right] \leq E \left[ \|z\|^2 \right] = \omega E \left[ \|z\|^2 \right].$$

The second inequality can be proved exactly same way. □

**Lemma 8 (Local quantization variance).** Suppose $x$ is fixed, $C \in B(\omega)$, and $g_i(x)$ is an unbiased estimator of $\nabla f_i(x)$. Then

$$E \left[ \|C(g_i(x)) - \nabla f_i(x)\|^2 \right] \leq (\omega + 1) \sigma_i^2 + \omega \|\nabla f_i(x)\|^2. \quad (18)$$

**Proof.**

$$E \left[ \|C(g_i(x)) - \nabla f_i(x)\|^2 \right] \text{ Def. 4+ } E \left[ \|C(g_i(x)) - g_i(x)\|^2 \right] + E \left[ \|g_i(x) - \nabla f_i(x)\|^2 \right] \leq \omega E \left[ \|g_i(x)\|^2 \right] + E \left[ \|g_i(x) - \nabla f_i(x)\|^2 \right] \text{ Def. 4+ } (\omega + 1)E \left[ \|g_i(x)\|^2 \right] + \omega \|\nabla f_i(x)\|^2 \text{ Assum. } \leq (\omega + 1) \sigma_i^2 + \omega \|\nabla f_i(x)\|^2. \quad \square$$

**Lemma 9 (Global quantization variance).** Suppose $x$ is fixed, $C_{W_i} \in B(\omega_{W_i})$ for all $i$, $C_M \in B(\omega_M)$, and $g_i(x)$ is an unbiased estimator of $\nabla f_i(x)$ for all $i$. Then

$$E \left[ \left\| \frac{1}{n} C_M \left( \sum_{i=1}^n C_{W_i}(g_i(x)) \right) \right\|^2 \right] \leq \alpha \|\nabla f(x)\|^2, \quad (19)$$

where $\omega_W = \max_{i \in [n]} \omega_{W_i}$ and

$$\alpha = \frac{\omega_M + 1}{n} \sigma^2 + \frac{\omega_M + 1}{n} \omega_{W} \zeta^2, \quad \beta = 1 + \omega_M + \frac{\omega_M + 1}{n} \omega_{W}. \quad (20)$$

**Proof.** For added clarity, let us denote

$$\Delta = \sum_{i=1}^n C_{W_i}(g_i(x)).$$
Using this notation, the proof proceeds as follows:

\[
\begin{align*}
\mathbb{E} \left[ \left\| \frac{1}{n} \mathcal{C}_M(\Delta) \right\|^2 \right] & \overset{\text{Def.} \ (17)}{=} \mathbb{E} \left[ \left\| \frac{1}{n} \mathcal{C}_M(\Delta) - \nabla f(x) \right\|^2 \right] + \left\| \nabla f(x) \right\|^2 \\
\overset{\text{Def.} \ (14)}{=} & \frac{1}{n^2} \mathbb{E} \left[ \| \mathcal{C}_M(\Delta) - \Delta \|^2 \right] + \mathbb{E} \left[ \left\| \frac{1}{n} \Delta - \nabla f(x) \right\|^2 \right] + \left\| \nabla f(x) \right\|^2 \\
\overset{(16)}{\leq} & \frac{\omega_M}{n^2} \mathbb{E} \left[ \| \Delta \|^2 \right] + \mathbb{E} \left[ \left\| \frac{1}{n} \Delta - \nabla f(x) \right\|^2 \right] + \left\| \nabla f(x) \right\|^2 \\
\overset{(13)}{\leq} & \frac{\omega_M + 1}{n} \mathbb{E} \left[ \left\| \nabla f(x) \right\|^2 \right] + \frac{(\omega_M + 1) \| \nabla f(x) \|^2}{n} \\
\overset{\text{Assum.} \ (2)}{\leq} & \frac{(\omega_M + 1)(\omega_W + 1)\sigma^2}{n} + \frac{(\omega_M + 1)\omega_W}{n} \sum_{i=1}^{n} \left\| \nabla f_i(x) - \nabla f(x) \right\|^2 \\
& + \left( 1 + \omega_M + \frac{(\omega_M + 1)\omega_W}{n} \right) \left\| \nabla f(x) \right\|^2 \\
\end{align*}
\]

\[\square\]

### D.4 Proof of Theorem \(6\)

Using \(L\)-smoothness of \(f\) and then applying Lemma \(9\) we get

\[
\begin{align*}
\mathbb{E} \left[ f(x^{k+1}) \right] & \leq \mathbb{E} \left[ f(x^k) \right] + \mathbb{E} \left[ \left\| \nabla f(x^k), x^{k+1} - x^k \right\|^2 \right] + \frac{L}{2} \mathbb{E} \left[ \left\| x^{k+1} - x^k \right\|^2 \right] \\
& \leq \mathbb{E} \left[ f(x^k) \right] - \eta_k \mathbb{E} \left[ \left\| \nabla f(x^k) \right\|^2 \right] + \frac{L}{2} \eta_k^2 \mathbb{E} \left[ \left\| g_k \right\|^2 \right] \\
& \overset{(19)}{=} \mathbb{E} \left[ f(x^k) \right] - \left( \eta_k - \frac{L}{2} \beta \eta_k^2 \right) \mathbb{E} \left[ \left\| \nabla f(x^k) \right\|^2 \right] + \frac{L}{2} \alpha \eta_k^2 .
\end{align*}
\]

Summing these inequalities for \(k = 0, ..., T - 1\), we obtain

\[
\sum_{k=0}^{T-1} \left( \eta_k - \frac{L}{2} \beta \eta_k^2 \right) \mathbb{E} \left[ \left\| \nabla f(x^k) \right\|^2 \right] \leq f(x^0) - f(x^*) + \frac{TL\alpha \eta_k^2}{2} .
\]

Taking \(\eta_k = \eta\) and assuming

\[
\eta \leq \frac{2}{L\beta} ,
\]

one obtains

\[
\mathbb{E} \left[ \left\| \nabla f(x^*) \right\|^2 \right] \leq \frac{1}{T} \sum_{k=0}^{T-1} \mathbb{E} \left[ \left\| \nabla f(x^k) \right\|^2 \right] \leq \frac{2(f(x^0) - f(x^*))}{T\eta(2 - L\beta \eta)} + \frac{L\alpha \eta}{2 - L\beta \eta} := \delta(\eta, T) .
\]

It is easy to check that if we choose \(\eta = \frac{\varepsilon}{L(\alpha + \varepsilon \beta)}\) (which satisfies \(21\) for every \(\varepsilon > 0\), then for any \(T \geq \frac{2L(f(x^0) - f(x^*))/(\alpha + \varepsilon \beta)}{\varepsilon^2}\) we have \(\delta(\eta, T) \leq \varepsilon\), concluding the proof.

\[25\]
D.5 A Different Stepsize Rule for Theorem 6

Looking at Theorem 6 one can see that setting step size
\[ \eta_k = \eta = \sqrt{\frac{2(f(x^0) - f(x^*))}{LT\alpha}} \]
with
\[ T \geq \frac{L\beta^2(f(x^0) - f(x^*))}{\alpha} \]
(number of iterations), we have iteration complexity
\[ \mathcal{O} \left( \sqrt{\frac{(\omega_W + 1)(\omega_M + 1)}{Tn}} \right), \]
which will be essentially the same as doing no compression on master and using \( \mathcal{C}_W \circ \mathcal{C}_M \) or \( \mathcal{C}_W \circ \mathcal{C}_M \) on the workers’ side. Our rate generalizes to the rate of [12] without compression and dependency on the compression operator is better comparing to the linear one in [23]. Moreover, our rate enjoys linear speed-up in the number of workers \( n \), the same as [12]. In addition, if one introduces mini-batching on each worker of size \( b \) and assuming each worker has access to the whole data, then \( \sigma^2 \to \sigma^2/b \) and \( \zeta^2 \to 0 \), which implies
\[ \mathcal{O} \left( \sqrt{\frac{(\omega_W + 1)(\omega_M + 1)}{Tbn}} \right), \]
and hence one can also obtain linear speed-up in terms of mini-batch size, which matches with [23].

D.6 SGD with Bidirectional Compression: Four Models

It is possible to consider several different regimes for our distributed optimization/training setup, depending on factors such as:

- The relative speed of communication (per bit) from workers to the master and from the master to the workers,
- The intelligence of the master, i.e., its ability or the lack thereof of the master to perform aggregation of real numbers (e.g., a switch can only perform integer aggregation),
- Variability of various resources (speed, memory, etc) among the workers.

For simplicity, we will consider four situations/ regimes only, summarized in Table 2.

**Table 2: Four theoretical models.**

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Model 1 (infinite fast communication)</th>
<th>Model 2 (switchML)</th>
<th>Model 3 (real numbers)</th>
<th>Model 4 (integer only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master can communicate infinitely fast</td>
<td>Master can aggregate real numbers (e.g., a workstation)</td>
<td>Master can aggregate integers only (e.g., SwitchML)</td>
<td>Master can aggregate real numbers (e.g., a workstation)</td>
<td>Master can aggregate integers only (e.g., SwitchML)</td>
</tr>
</tbody>
</table>

**D.5 A Different Stepsize Rule for Theorem 6**

Looking at Theorem 6 one can see that setting step size
\[ \eta_k = \eta = \sqrt{\frac{2(f(x^0) - f(x^*))}{LT\alpha}} \]
with
\[ T \geq \frac{L\beta^2(f(x^0) - f(x^*))}{\alpha} \]
(number of iterations), we have iteration complexity
\[ \mathcal{O} \left( \sqrt{\frac{(\omega_W + 1)(\omega_M + 1)}{Tn}} \right), \]
which will be essentially the same as doing no compression on master and using \( \mathcal{C}_W \circ \mathcal{C}_M \) or \( \mathcal{C}_W \circ \mathcal{C}_M \) on the workers’ side. Our rate generalizes to the rate of [12] without compression and dependency on the compression operator is better comparing to the linear one in [23]. Moreover, our rate enjoys linear speed-up in the number of workers \( n \), the same as [12]. In addition, if one introduces mini-batching on each worker of size \( b \) and assuming each worker has access to the whole data, then \( \sigma^2 \to \sigma^2/b \) and \( \zeta^2 \to 0 \), which implies
\[ \mathcal{O} \left( \sqrt{\frac{(\omega_W + 1)(\omega_M + 1)}{Tbn}} \right), \]
and hence one can also obtain linear speed-up in terms of mini-batch size, which matches with [23].
The upper bound is achieved for \( n = 1 \) (or for any \( n \) and \( \varepsilon \to 0 \)), and the lower bound is achieved in the limit as \( n \to \infty \). So, the slowdown caused by compression on worker side decreases with \( n \). More importantly, the savings in communication due to compression can outweigh the iteration slowdown, which leads to an overall speedup!

### D.6.1 Model 1

First, we start with the comparison, where we assume that transmitting one bit from worker to node takes the same amount of time as from master to worker.

<table>
<thead>
<tr>
<th>Approach</th>
<th>( C_W )</th>
<th>No. iterations ( T(\omega) = \mathcal{O}((\omega + 1)^{1+\theta}) )</th>
<th>Bits per iteration ( W_i \mapsto M + M \mapsto W_i )</th>
<th>Speedup ( \frac{T(\omega)<em>{\text{bits}}}{T(\omega)</em>{\text{std}}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>identity</td>
<td>( 1 )</td>
<td>( 2 \cdot 32d )</td>
<td>( \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
<tr>
<td>New</td>
<td>( C_{\text{nat}} )</td>
<td>( \left(\frac{9}{8}\right)^{1+\theta} )</td>
<td>( 2 \cdot 9d )</td>
<td>( 2.81 \times \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
<tr>
<td>Sparsification</td>
<td>( S^q )</td>
<td>( \left(\frac{9}{8}\right)^{1+\theta} )</td>
<td>( 2 \cdot (33 + \log_2 d)q )</td>
<td>( 0.66 \times \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
<tr>
<td>New</td>
<td>( C_{\text{nat}} \circ S^q )</td>
<td>( \left(\frac{9}{8}\right)^{1+\theta} )</td>
<td>( 2 \cdot (10 + \log_2 d)q )</td>
<td>( 0.09 \times \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
<tr>
<td>Dithering</td>
<td>( D_{\text{sta}}^{p,\tau-1} )</td>
<td>( \left(1 + \sqrt{2} \sqrt{2}/\kappa\right)^{1+\theta} )</td>
<td>( 2 \cdot (32 + d(s + 2)) )</td>
<td>( 1.67 \times \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
<tr>
<td>New</td>
<td>( D_{\text{sta}}^{p,s} )</td>
<td>( \left(\frac{81}{64} + \frac{1}{2} \sqrt{2} \sqrt{2}/\kappa\right)^{1+\theta} )</td>
<td>( 2 \cdot (8 + d(\log_2 s + 2)) )</td>
<td>( 3.19 \times \frac{T(\omega)<em>{\text{std}}}{T(\omega)</em>{\text{bits}}} )</td>
</tr>
</tbody>
</table>

**Table 3**: Our compression techniques can speed up the overall runtime (number of iterations \( T(\omega) \) times the bits sent per iteration) of distributed SGD. We assume binary32 floating point representation, bi-directional compression using \( C \), and the same speed of communication from worker to master \((W_i \mapsto M)\) and back \((M \mapsto W_i)\). The relative number of iterations (communications) sufficient to guarantee \( \varepsilon \) optimality is \( T(\omega) := (\omega + 1)^{\theta} \), where \( \theta \in (1, 2) \) (see Theorem 6). Note that big \( n \) regime leads to better iteration bound \( T(\omega) \) since for big \( n \) we have \( \theta \approx 1 \), while for small \( n \) we have \( \theta \approx 2 \). For dithering, \( \kappa = \min\{1, \sqrt{2}\} \). The speedup for \( C_{\text{nat}} \) is obtained for \( \theta = 1 \), and the speedup for \( \theta = 0 \). The speedup figures were calculated for \( d = 10^6 \), \( p = 2 \) (dithering), optimal choice of \( s \) (dithering), and \( q = 0.1d \) (sparsification).

### D.6.2 Model 2

For the second model, we assume that the master communicates much faster than workers thus communication from workers is the bottleneck and we don’t need to compress updates after aggregation, thus \( C_M \) is identity operator with \( \omega_M = 0 \). This is the case we mention in the main paper. For completeness, we provide the same table here.

<table>
<thead>
<tr>
<th>Approach</th>
<th>( C_W )</th>
<th>No. iterations ( T'(\omega_W) = \mathcal{O}(\omega_W + 1)^{\theta} )</th>
<th>Bits per 1 iter. ( W_i \mapsto M )</th>
<th>Speedup Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>identity</td>
<td>( 1 )</td>
<td>( 32d )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>New</td>
<td>( C_{\text{nat}} )</td>
<td>( (\gamma/s)^\theta )</td>
<td>( 9d )</td>
<td>( 3.2 \times 3.6 \times )</td>
</tr>
<tr>
<td>Sparsification</td>
<td>( S^q )</td>
<td>( (\gamma/s)^\theta )</td>
<td>( (33 + \log_2 d)q )</td>
<td>( 0.6 \times 6.0 \times )</td>
</tr>
<tr>
<td>New</td>
<td>( C_{\text{nat}} \circ S^q )</td>
<td>( (\gamma/s)^\theta )</td>
<td>( (10 + \log_2 d)q )</td>
<td>( 1.0 \times 10.7 \times )</td>
</tr>
<tr>
<td>Dithering</td>
<td>( D_{\text{sta}}^{p,\tau-1} )</td>
<td>( (\gamma^2/s^\theta \gamma)^\theta )</td>
<td>( 31 + d(2 + s) )</td>
<td>( 1.8 \times 15.9 \times )</td>
</tr>
<tr>
<td>New</td>
<td>( D_{\text{sta}}^{p,s} )</td>
<td>( (\gamma^2/s^\theta \gamma^2)^\theta )</td>
<td>( 8 + d(2 + \log_2 s) )</td>
<td>( 4.1 \times 16.0 \times )</td>
</tr>
</tbody>
</table>

**Table 4**: The overall speedup of distributed SGD with compression on nodes via \( C_W \) over a Baseline variant without compression. Speed is measured by multiplying the # communication rounds (i.e., iterations \( T(\omega_W) \)) by the bits sent from worker to master \((W_i \mapsto M)\) per 1 iteration. We neglect \( M \mapsto W_i \) communication as in practice this is much faster. We assume binary32 representation. The relative # iterations sufficient to guarantee \( \varepsilon \) optimality is \( T'(\omega_W) := (\omega_W + 1)^{\theta} \), where \( \theta \in (0, 1] \) (see Theorem 6). Note that in the big \( n \) regime the iteration bound \( T(\omega_W) \) is better due to \( \theta \approx 0 \) (however, this is not very practical as \( n \) is usually small), while for small \( n \) we have \( \theta \approx 1 \). For dithering, \( r = \min\{p, 2\} \), \( \kappa = \min\{1, \sqrt{2}\} \). The lower bound for the Speedup Factor is obtained for \( \theta = 1 \), and the upper bound for \( \theta = 0 \). The Speedup Factor \( \frac{T(\omega_W) \# \text{Bits}}{T(\omega_W)_{\text{std}} \# \text{Bits}} \) figures were calculated for \( d = 10^6 \), \( q = 0.1d \), \( p = 2 \) and optimal choice of \( s \) with respect to speedup.
D.6.3 Model 3

Similarly to previous sections, we also do the comparison for methods that might be used for In-Network Aggregation. Note that for INA, it is useful to do compression also from master back to workers as the master works just with integers, hence in order to be compatible with floats, it needs to use bigger integers format. Moreover, $C_{\text{nat}}$ compression guarantees free translation to floats. For the third model, we assume we have the same assumptions on communication as for Model 1. As a baseline, we take SGD with $C_{\text{nat}}$ as this is the most simple analyzable method, which supports INA.

<table>
<thead>
<tr>
<th>Approach</th>
<th>$C$</th>
<th>Slowdown (iters / baseline)</th>
<th>Bits per iter. $W_i \mapsto M + M \mapsto W_i$</th>
<th>Speedup factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>$C_{\text{nat}}$</td>
<td>$\frac{1}{2}$</td>
<td>$2 \cdot 9d$</td>
<td>$1$</td>
</tr>
<tr>
<td>Sparsification</td>
<td>$S^y \circ C_{\text{nat}}$</td>
<td>$(\frac{q}{y})^{1+s}$</td>
<td>$2 \cdot (10 + \log_2 d)q$</td>
<td>$0.03 \times 0.30 \times$</td>
</tr>
<tr>
<td>Dithering</td>
<td>$D^{p,s}_{\text{nat}}$</td>
<td>$(\frac{q}{s} + \kappa d + 2^{1-s})^{1+\theta}$</td>
<td>$2 \cdot (8 + d(2 + \log_2 s))$</td>
<td>$1.14 \times 1.30 \times$</td>
</tr>
</tbody>
</table>

Table 5: Overall speedup (number of iterations $T$ times the bits sent per iteration ($W_i \mapsto M + M \mapsto W_i$) of distributed SGD. We assume binary32 floating point representation, bi-directional compression using the same compression $C$. The relative number of iterations (communications) sufficient to guarantee $\varepsilon$ optimality is displayed in the third column, where $\theta \in (0, 1]$ (see Theorem 3). Note that big $n$ regime leads to smaller slowdown since for big $n$ we have $\theta \approx 0$, while for small $n$ we have $\theta \approx 1$. For dithering, we chose $p = 2$ and $\kappa = \min\{1, \sqrt{d^{1-s}}\}$. The speedup factor figures were calculated for $d = 10^6$, $p = 2$ (dithering), optimal choice of $s$ (dithering), and $q = 0.1d$ (sparsification).

D.6.4 Model 4

Here, we do the same comparison as for Model 3. In contrast, for communication we use the same assumptions as for Model 2.

<table>
<thead>
<tr>
<th>Approach</th>
<th>$C_{W_i}$</th>
<th>$C_{M}$</th>
<th>Slowdown (iters / baseline)</th>
<th>$W_i \mapsto M$ commun. (bits / iteration)</th>
<th>Speedup factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>$C_{\text{nat}}$</td>
<td>$C_{\text{nat}}$</td>
<td>$\frac{1}{2}$</td>
<td>$9d$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>Sparsification</td>
<td>$S^y \circ C_{\text{nat}}$</td>
<td>$C_{\text{nat}}$</td>
<td>$(\frac{q}{y})^{s}$</td>
<td>$(10 + \log_2 d)q$</td>
<td>$0.30 \times 3 \times$</td>
</tr>
<tr>
<td>Dithering</td>
<td>$D^{p,s}_{\text{nat}}$</td>
<td>$C_{\text{nat}}$</td>
<td>$(\frac{q}{s} + \kappa d + 2^{1-s})^{\theta}$</td>
<td>$(8 + d(2 + \log_2 s))$</td>
<td>$1.3 \times 4.5 \times$</td>
</tr>
</tbody>
</table>

Table 6: Overall speedup (number of iterations $T$ times the bits sent per iteration ($W_i \mapsto M$) of distributed SGD. We assume binary32 floating point representation, bi-directional compression using $C_{W_i}, C_M$. The relative number of iterations (communications) sufficient to guarantee $\varepsilon$ optimality is displayed in the third column, where $\theta \in (0, 1]$ (see Theorem 3). Note that big $n$ regime leads to smaller slowdown since for big $n$ we have $\theta \approx 0$, while for small $n$ we have $\theta \approx 1$. For dithering, we chose $p = 2$ and $\kappa = \min\{1, \sqrt{d^{1-s}}\}$. The speedup factor figures were calculated for $d = 10^6$, $p = 2$ (dithering), optimal choice of $s$ (dithering), and $q = 0.1d$ (sparsification).

D.6.5 Communication strategies used in Tables

No Compression or $C_{\text{nat}}$. Each worker has to communicate a (possibly dense) $d$ dimensional vector of scalars, each represented by 32 or 9 bits, respectively.

Sparsification $S^y$ with or without $C_{\text{nat}}$. Each worker has to communicate a sparse vector of $q$ entries with full 32 or limited 9 bit precision. We assume that $q$ is small, hence one would prefer to transmit positions of non-zeros, which takes $q(\log_2 (d) + 1)$ additional bits for each worker.

Dithering ($D^{p,s}_{\text{nat}}$ or $D^{p,s}_{\text{nat}}$). Each worker has to communicate $31(8 - D^{p,s}_{\text{nat}})$ bits (sign is always positive, so does not need to be communicated) for the norm, and $\log_2 (s) + 1$ bits for every coordinate for level encoding (assuming uniform encoding) and 1 bit for the sign.

D.7 Sparsification - Formal Definition

Here we give a formal definition of the sparsification operator $S^y$ used in Tables.

28
Definition 5 (Random sparsification). Let \( 1 \leq q \leq d \) be an integer, and let \( \circ \) denote the Hadamard (element-wise) product. The random sparsification operator \( S^q : \mathbb{R}^d \to \mathbb{R}^d \) is defined as follows:

\[
S^q(x) = \frac{d}{q} \cdot \xi \circ x,
\]

where \( \xi \in \mathbb{R}^d \) is a random vector chosen uniformly from the collection of all binary vectors \( y \in \{0, 1\}^d \) with exactly \( q \) nonzero entries (i.e., \( \|y\|_0 = q \)).

The next result describes the variance of \( S^q \):

Theorem 10. \( S^q \in \mathbb{B}(d/q - 1) \).

Notice that in the special case \( q = d \), \( S^q \) reduces to the identity operator (i.e., no compression is applied), and Theorem 10 yields a tight variance estimate: \( d/d - 1 = 0 \).

Proof. See e.g. \cite[Lemma A.1]{44}.

Let us now compute the variance of the composition \( C_{\text{nat}} \circ S^q \). Since \( C_{\text{nat}} \in \mathbb{B}(1/8) \) (Theorem 1) and \( S^q \in \mathbb{B}(d/q - 1) \) (Theorem 10), in view of the our composition result (Theorem 3) we have

\[
C_W = C_{\text{nat}} \circ S^q \in \mathbb{B}(\omega_W), \quad \text{where} \quad \omega_W = \frac{1}{8} \left( \frac{d}{q} - 1 \right) + \frac{1}{8} + \frac{d}{q} - 1 = \frac{9d}{8q} - 1. \tag{23}
\]

E Limitations and Extensions

Quantization techniques can be divided into two categories: biased \cite{4, 44} and unbiased \cite{2, 48, 47}. While the focus of this paper was on unbiased quantizations, it is possible to combine our natural quantization mechanisms in conjunction with biased techniques, such as the TopK sparsifier proposed in \cite{9, 1} and recently analyzed in \cite{4, 44}, and still obtain convergence guarantees.