An Efficient Statistical-based Gradient Compression Technique for Distributed Training Systems

Ahmed M. Abdelmoniem [* 1] **Ahmed Elzanaty** [* 1] **Mohamed-Slim Alouini** [1] **Marco Canini** [1]

Abstract

The recent many-fold increase in the size of deep neural networks makes efficient distributed training challenging. Many proposals exploit the compressibility of the gradients and propose lossy compression techniques to speed up the communication stage of distributed training. Nevertheless, compression comes at the cost of reduced model quality and extra computation overhead. In this work, we design an efficient compressor with minimal overhead. Noting the sparsity of the gradients, we propose to model the gradients as random variables distributed according to some sparsity-inducing distributions (SIDs). We empirically validate our assumption by studying the statistical characteristics of the evolution of gradient vectors over the training process. We then propose *Sparsity-Inducing Distribution-based Compression (SIDCo)*, a threshold-based sparsification scheme that enjoys similar threshold estimation quality to deep gradient compression (DGC) while being faster by imposing lower compression overhead. Our extensive evaluation of popular machine learning benchmarks involving both recurrent neural network (RNN) and convolution neural network (CNN) models shows that *SIDCo* speeds up training by up to $\approx 41.7\times$, $7.6\times$, and $1.9\times$ compared to the no-compression baseline, Top$_k$, and DGC compressors, respectively.

1 Introduction

As deep neural networks (DNNs) continue to become larger and more sophisticated, and ever increasing amounts of training data are used (Brown et al., 2020; Shoeybi et al., 2019), scaling the training process to run efficiently on a distributed cluster is currently a crucial objective that is attracting a multitude of efforts (Kurth et al., 2018; Narayanan et al., 2019; Peng et al., 2019; Verbraeken et al., 2020). Modern deep learning toolkits (Abadi et al., 2016; pytorch.org; Sergeev & Balso, 2018) are capable of distributed data-parallel training whereby the model is replicated and training data are partitioned among workers. Training DNNs in such settings in practice relies on synchronous distributed Stochastic Gradient Descent (SGD) or similar optimizers (refer to Appendix A for more details). Let N be the number of workers, and $\mathbf{x}_{\{i\}} \in \mathbb{R}^d$ denote the model parameters with d dimensions at iteration i. At the end of the i^{th} iteration, each worker runs the back-propagation algorithm to produce a local stochastic gradient, $\mathbf{g}^n_{\{i\}} \in \mathbb{R}^d$, at worker n. Then, each worker updates its model parameters using the final gradient aggregated across all workers as $\mathbf{x}_{\{i+1\}} = \mathbf{x}_{\{i\}} - \lambda \frac{1}{N}\sum_{n=1}^{N} \mathbf{g}^n_{\{i\}}$, where λ is the learning rate. Gradient aggregation involves communication, which is either between the workers in a peer-to-peer fashion (typically through collective communication primitives like all-reduce) or via a parameter server architecture. Due to the synchronous nature of the optimizer, workers cannot proceed with the $(i+1)^{\text{th}}$ iteration until the aggregated gradient is available. Therefore, in distributed training workloads, communication is commonly one of the predominant bottlenecks (Fang et al., 2019; Lin et al., 2018).

Addressing this communication bottleneck is the focus of this paper, where we pursue the path of improving training by reducing the communicated data volume via lossy gradient compression. Compression entails two main challenges: (i) it can negatively affect the training accuracy (because the greater the compression is, the larger the error in the aggregated gradient), and (ii) it introduces extra computation latency (due to the compression operation itself). While the former can be mitigated by applying compression to a smaller extent or using compression with error-feedback (Karimireddy et al., 2019; Lin et al., 2018), the latter, if gone unchecked, can actually slow down training compared to not compressing. Surprisingly, much of the prior works in this area ignored the computation overheads of compression. Given that modern clusters for deep learning workloads nowadays use high speed, low latency network fabrics (e.g., 100 Gbps Ethernet or InfiniBand), we argue that the efficiency of compression needs to be explicitly accounted for.

Motivated by the above observations, we propose *SIDCo*

[*]Equal contribution [1]KAUST. Correspondence to: Marco Canini <marco@kaust.edu.sa>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021. Copyright 2021 by the author(s).

compression.[1] *SIDCo* builds on a sound theory of signal compressibility and enjoys linear complexity in the size of model parameters. Importantly, this affords for an implementation that parallelizes very efficiently using modern GPUs and other hardware targets. Thus, our work addresses a previously-overlooked yet crucial technical obstacle to using compression in practice, especially for communication-bounded training of large models.

1.1 Related Work

Efficient communication in distributed training has received extensive attention (Narayanan et al., 2019; Wangni et al., 2018; Xu et al., 2020). One approach tries to maximize the overlap between the computation and communication to hide the communication overhead (Narayanan et al., 2019; Peng et al., 2019). However, the gains from these methods are bounded by the length of computation and are modest when the training is dominantly communication-bound. Alternatively, many approaches adopt methods that reduce the amount of communication, volume (Lin et al., 2018) or frequency (Dieuleveut & Patel, 2019). In this work, we focus on gradient compression as it shows considerable benefits.

Gradient Compression is a well-known volume reduction technique (Dutta et al., 2020; Fang et al., 2019; Lin et al., 2018; Wangni et al., 2018; Xu et al., 2020). Each worker applies a compression operator $\mathbb{C}$ to $\mathbf{g}^n_{\{i\}}$ to produce a compressed gradient vector that is transmitted for aggregation. Generally, the compressor $\mathbb{C}$ involves quantization and/or sparsification operations.

Gradient Quantization represents gradients with fewer bits for each gradient element. Under some conditions, quantization is known to achieve the same convergence as no compression (Fu et al., 2020; Wu et al., 2018). Error compensation (EC) is used to attain convergence when gradients are quantized using fewer bits (Karimireddy et al., 2019; Wu et al., 2018; Xu et al., 2020). Given the standard 32-bit float number representation, the volume reduction of quantization is limited by $32\times$, i.e., 1 bit out of 32 bits, which may not be sufficient for large models or slow networks and it requires expensive encoding to pack the quantized bits (Gajjala et al., 2020).

Gradient Sparsification selects a subset of gradient elements. It is generally more flexible than quantization, as it can reduce volume by up to $d\times$ and adapts easily to network conditions (Abdelmoniem & Canini, 2021). It was shown that in some cases, up to 99.9% of the non-significant gradient elements can be dropped with limited impact on convergence (Aji & Heafield, 2017; Lin et al., 2018; Shi et al., 2019). Gradient sparsification using Top$_k$ – selecting the top k elements by their magnitude – is known to yield better convergence compared to other compression schemes, e.g., Random-k (Alistarh et al., 2018; Lin et al., 2018). However, Top$_k$ or its variants are notorious for being computationally inefficient (Xu et al., 2020). Top$_k$ selection does not perform well on accelerators such as GPUs (Shanbhag et al., 2018). For instance, in many cases, it is reported that Top$_k$ imposes high overheads and worsens the run-time of distributed training (Shi et al., 2019; Xu et al., 2020).

1.2 Background and Motivation

The main challenge with using gradient compression (e.g., sparsification or quantization) is the computational overhead it introduces in the training. If the overhead is greater than the reduction gains in communication time, the overall iteration time increases. Hence, to be useful, a robust compressor should have a low overhead (Fang et al., 2019; Shi et al., 2019). As presented earlier, one of the dominantly robust compressors is Top$_k$, however it is also computationally heavy (Fang et al., 2019; Shi et al., 2019; 2020). Because of this, Top$_k$, for large models, results in either an increased training time or unsatisfactory performance benefits.

Numerous efforts based on algorithmic or heuristic approaches have been dedicated to enhancing the performance of Top$_k$ (Jiang et al., 2018; Lin et al., 2018; Shanbhag et al., 2018; Shi et al., 2019). Existing fast implementations of Top$_k$ are compute-intensive (e.g., on CPU, the computational complexity is $\mathcal{O}(d\ \log_2 k)$) (Shanbhag et al., 2018). Recently, more optimized implementations for multi-core hardware are proposed, which greatly depend on the data distribution and work best for a small number of k (Shanbhag et al., 2018). For instance, the Radix select algorithm used in PyTorch is $\mathcal{O}\left(\lceil b/r \rceil\ d\right)$ where b is the number of bits in the data values and r is the radix size (pytorch.org). Yet, using gradient vectors of various sizes, Top$_k$ is the slowest on GPUs and not the fastest on CPUs, as shown later in our micro-benchmark and in Appendix E.2.

In the context of gradient compression, *Threshold-based methods*, aiming to overcome the overhead of Top$_k$, select in linear time gradient elements larger in magnitude than a threshold η. DGC (Lin et al., 2018) proposes to sample a random subset of the gradients (e.g., 1%), apply Top$_k$ on the sampled sub-population to find a threshold which is then used to obtain the actual Top$_k$ elements hierarchically.[2] Even though DGC leads to improved performance over Top$_k$, its computational complexity is still in the same order of Top$_k$'s complexity. *Threshold estimation methods* on the other hand, are shown to attain linear time complexity (Aji

[1]Our code release is available at `https://github.com/sands-lab/SIDCo`.

[2]Aside from the expensive random sampling, in worst case, DGC invokes Top$_k$ twice, once on the subset to obtain a threshold and another to obtain k elements if the number of elements obtained via the threshold are more than the target k.

& Heafield, 2017; Alistarh et al., 2018; Dryden et al., 2016).

Recently, several works have leveraged certain features of the gradients to enhance the training process (Fu et al., 2020; Narang et al., 2018). Some approaches leveraged these features and devised heuristics to estimate and find the Top$_k$ threshold which exhibits lower compression overhead compared to Top$_k$ and DGC (Fang et al., 2019; Shi et al., 2019). In particular, RedSync (Fang et al., 2019) finds the threshold by moving the ratio between the maximum and mean values of the gradient; GaussianKSGD (Shi et al., 2019) adjusts an initial threshold obtained from fitting a Gaussian distribution through an iterative heuristic to obtain the Top$_k$ elements. Nevertheless, the threshold estimation of these methods is of bad quality and the number of selected elements, $\hat{k}$, varies significantly from the target k (Section 4.1).

In this work, we propose a statistical approach to estimate an accurate threshold for selecting the Top$_k$ elements with minimal overhead. In particular, we exploit the compressibility of the gradients and opt for SIDs that fit the gradients well. For instance, double exponential (i.e., Laplace), double gamma and double generalized Pareto distributions have been used as sparsity-promoting priors in Bayesian estimation framework (Armagan et al., 2013; Babacan et al., 2010; Monga et al., 2018). Our study of the gradients supports the assumption for their compressibility and suitability for modeling the gradients as random variables (r.v.s) distributed according to one of the SIDs.

To motivate our approach, we conduct initial micro-benchmark experiments to evaluate the compression overhead of sparsification techniques: Top$_k$, DGC (which uses random sub-sample for threshold calculation), RedSync and GaussianKSGD (which heuristically estimate the threshold), and one of our proposed *SIDCo* schemes that estimates the threshold via a multi-stage fitting (Section 2). We use both CPU and GPU to benchmark the performance (see Appendix D). We show the speed-up of different compressors normalized by the compression speed of Top$_k$. We observe from the results that methods based on random sub-sampling (e.g., DGC) excel on GPU (Figure 1a), but they imposes huge overhead on CPU and leads to DGC performing significantly worse than Top$_k$ on CPU (Figure 1b). In contrast, methods that are based on estimating a threshold over which only k elements are selected, impose consistently lower compression overhead compared to Top$_k$ and DGC on both GPU and CPU. This shows that, except for linear time threshold-based methods, a variable compression overhead is to be expected on different architectures (e.g., CPU, GPU, TPU, FPGA or AI chips).[3] Figure 1c shows the normalized actual compression ratio (i.e., $\hat{k}/k$) for various schemes; note that the heuristic approaches fail to obtain the right threshold, leading to unpredictable behavior.

[3]We note that many efforts are dedicated to the optimization and enabling of fast training on low-cost devices such as CPUs instead of opting for expensive hardware accelerations (Beidi et al., 2020; Das et al., 2018; Vanhoucke et al., 2011).

1.3 Contributions

In this work, we make the following contributions:

- We exploit the sparsity of the gradients via modeling the gradients as r.v.s with SIDs and propose a multi-stage fitting technique based on peak over threshold (PoT) which works well with aggressive sparsification ratios and adapts to the distribution changes of the gradient.
- We design *SIDCo*, a threshold sparsification method with closed-form expressions for three SIDs to keep the compression overhead as low as possible.
- We show that *SIDCo* consistently outperforms existing approaches via an extensive set of numerical and experimental evaluation on different benchmarks.

2 Proposed Gradient Model and Threshold Estimation

We discuss the compressibility of the gradients and their statistical distribution. Then, two threshold-based schemes are proposed that leverage the compressibility of the gradients.

2.1 Gradient Compressibility

Signals, including gradient vectors of DNNs, can be efficiently compressed by exploiting some of their inherent features. Among these features, sparsity and compressibility are the key drivers for performing signal compression (Elzanaty et al., 2019a;b; Mallat, 2009).

We start by a precise definition of compressible signals.

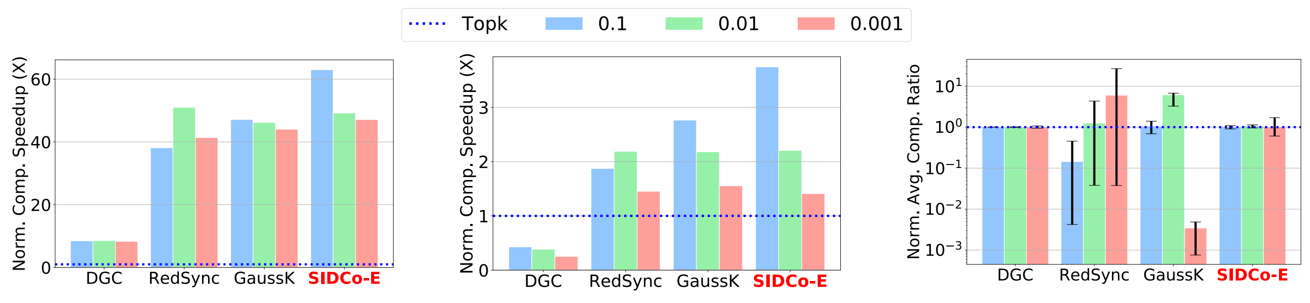
Definition 1 (Compressible Signals (Baraniuk et al., 2011)). *The signal $\mathbf{g} \in \mathbb{R}^d$ is compressible if the magnitudes of its sorted coefficients obey the following power law decay:*

$$\tilde{g}_j \le c_1 \, j^{-p} \quad \forall j \in \{1, 2, \cdots, d\}, \tag{1}$$

where $\tilde{\mathbf{g}}$ is the sorted vector of $|\mathbf{g}|$ in descending order, $\tilde{g}_j$ is the j^{th} element of $\tilde{\mathbf{g}}$, and $p > 1/2$ is the decay exponent, for some constant c_1. For compressible signals with power law decay, the sparsification error, $\sigma_k(\mathbf{g})$, is bounded as

$$\sigma_k(\mathbf{g}) \triangleq \|\mathbf{g} - \mathbb{T}_k\{\mathbf{g}\}\|_2 \le c_2 \, k^{1/2-p}, \tag{2}$$

where $\|\mathbf{x}\|_q = \left(\sum_{j=1}^{d} \mathbf{x}_j^q\right)^{1/q}$ is the ℓ_q norm of $\mathbf{x}$, $\mathbb{T}_k\{\cdot\}$ is the Top$_k$ sparsification operator that keeps only the largest k elements in magnitude and set the others to zero, $\mathbb{T}_k\{\mathbf{g}\}$ is a k-sparse vector with only k non-zero elements, and c_2 is a constant. The signal is more compressible if it decays faster, i.e., p is higher (DeVore, 1998).



(a) Compression with GPU (b) Compression with CPU (c) Quality of Threshold Estimation

Figure 1. The compression speedups over Top$_k$ using different compression ratios $(0.1, 0.01, 0.001)$, on (a) GPU and (b) CPU. (c) shows the average estimation quality of the target k. The experiments are performed for VGG16 (Table 1) with the setup detailed in §4.

Table 1. Summary of the benchmarks used in this work.

Task	Neural Network	Model	Dataset	Training Parameters	Per-Worker Batch Size	Learning Rate	Epochs	Comm Overhead	Local Optimizer	Quality metric
Language Modeling	RNN	LSTM (Hochreiter & Schmidhuber, 1997) 2 layers-1500 hidden units	PTB (Marcus et al., 1999)	66,034,000	20	22	30	94%	NesterovMom-SGD	Test Perplexity
Speech Recognition	RNN	LSTM 5 layers-1024 hidden units	AN4 (AN4)	43,476,256	20	0.004	150	80%	NesterovMom-SGD	WER & CER
Image Classification	CNN	ResNet-20 (He et al., 2015)	CIFAR-10 (Krizhevsky, 2009)	269,467	512	0.1	140	10%	SGD	Top-1 Accuracy
		VGG16 (Simonyan & Zisserman, 2015)	CIFAR-10	14,982,987	512	0.1	140	60%	SGD	
		ResNet-50	ImageNet (Deng et al., 2009)	25,559,081	160	0.2	90	72%	NesterovMom-SGD	
		VGG19	ImageNet	143,671,337	160	0.05	90	83%	NesterovMom-SGD	

Property 1 (Gradients Compressibility)**.** *The gradients generated during the training of most DNNs are compressible in the sense of Definition 1.*

Reasoning. From Definition 1, it can be verified whether the gradient vectors are compressible. In Appendix B.1, we empirically validate that the gradients generated during the training of widely-adopted DNNs respect the condition for compressibility stated in (1) and (2).

□

2.2 Gradient Modeling

The target now is to find the distribution of the gradient vector, while accounting for the compressibility of the gradients. The selection of sparsity-promoting priors that are able to efficiently capture the statistical characteristics of the gradients with low computational complexity is a challenging task. However, we notice an essential property for the distribution of the gradients that permits high compression gains with low computational overhead.

Property 2. *Gradients generated from many DNNs during the training can be modeled as r.v.s distributed according to some sparsity-inducing distributions, i.e., double exponential, double gamma and double generalized Pareto (GP) distributions. More precisely, we have*

$$G \dot{\sim} \text{Distribution}(\mathbf{\Theta}), \tag{3}$$

where Distribution$(\cdot)$ *is one of the three SIDs with parameters indicated by the vector* $\mathbf{\Theta}$ *that generally depends on the iteration and worker's data. Also, the probability density function (PDF) of* G*,* $f_G(g; \mathbf{\Theta})$*, is symmetric around zero.*

Reasoning. Since the gradients are compressible as indicated by Property 1, they can be well approximated by sparse vectors with minimal error, as implied from (2). Hence, the distributions that promote sparsity are good candidates for fitting (or modeling) the gradient vectors.[4] For instance, the double exponential, double gamma, double GP, and Bernoulli-Gaussian distributions have been used as priors that promote sparsity in (Armagan et al., 2013; Babacan et al., 2010; Elzanaty et al., 2019b; Monga et al., 2018). Property 2 is empirically verified for several DNN architectures and datasets in Section 4 and Appendix B.2.

For instance, we consider the gradients resulting from the training of ResNet-20 with SGD. The collected gradients are fitted by the three proposed SIDs, i.e., double exponential, double gamma, and double GP distributions. In Figure 2, the empirical distribution of the gradients and their absolutes, without EC mechanism, are shown along

[4]For threshold estimation, we are interested in the distribution of the amplitude of a random element in the gradient vector.

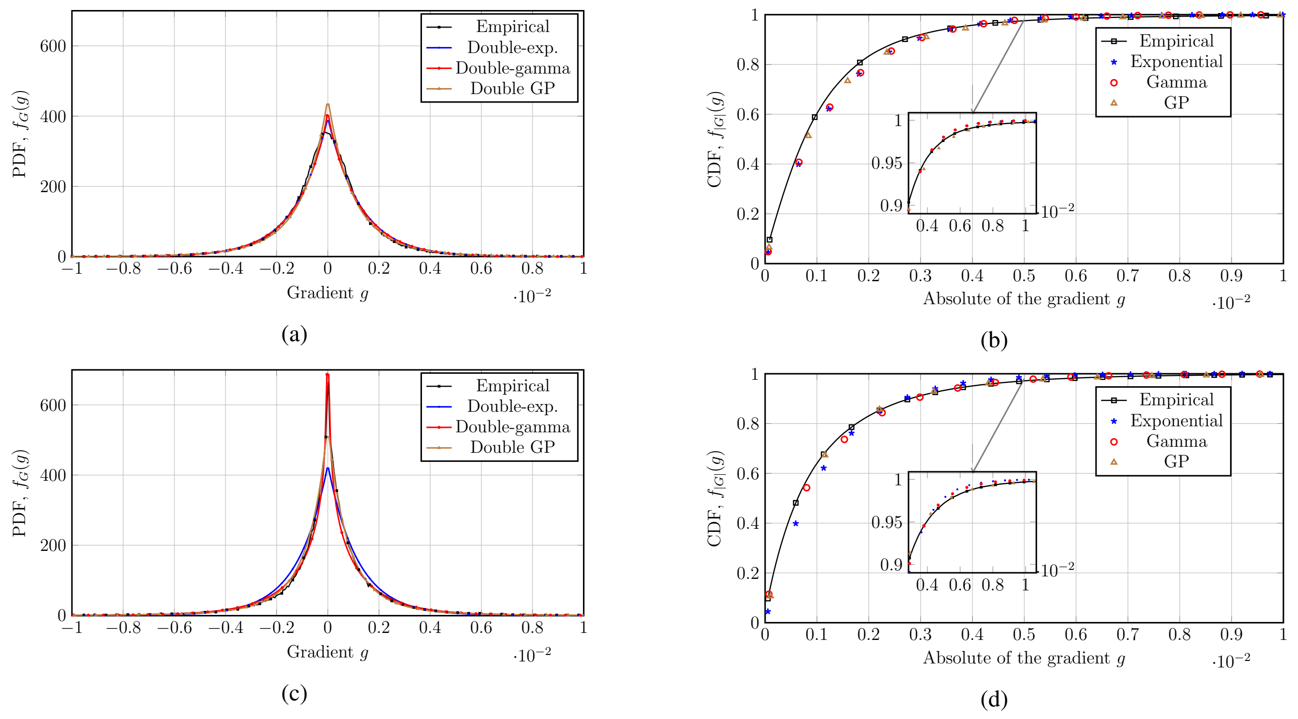


Figure 2. Fitting using the three SIDs for the gradient vector along with the empirical distribution generated from training ResNet-20 on CIFAR10 using Top$_k$ compressor without EC mechanism, for the 100th [(a) PDF, (b) CDF] and 10000th [(c) PDF, (d) CDF] iterations.

with the distributions of the three fitted SID for two iterations. We can notice in Figure 2a that the three proposed distributions can capture the main statistical characteristic of the gradients, as their PDFs approximate the empirical distribution for most of the gradient domain. This can be understood because of the compressibility of the gradients illustrated before. The compressibility of r.v.s distributed according to one of the SIDs can be attributed to the shape of their PDFs, where the most probable values are those with small amplitudes. From Figure 2a and Figure 2c, it can be seen that the gradients at iteration 10000 (Figure 2c) are more sparse than those at iteration 100 (Figure 2a), where the PDF at iteration 10000 has higher values at smaller gradient values and it has faster tail. Regarding the cumulative distribution function (CDF) of the absolute value of the gradients in Figure 2b and Figure 2d, we can see that the SIDs well approximate the empirical CDF. However, at the tail of the distribution, they tend to overestimate/underestimate the CDF slightly. The reason is that the fitting is biased toward the majority of the data with lower values, as the gradient vector is sparse. $\square$

2.3 Single-Stage Threshold Estimator

We now describe the proposed compression scheme. First, the threshold that yields the target compression ratio, $\delta \triangleq k/d$, is derived for each of the three proposed SIDs. Then, we present a single-stage thresholding scheme for moderate compression ratios. For aggressive compression ratios with $\delta \ll 1$, e.g., $\delta \leq 0.001$, we propose a multi-stage thresholding scheme to accurately estimate the threshold. The sparsification threshold can be computed from the fitted distribution of the gradients as follows:

Lemma 1. *For $G \sim$ Distribution$(\boldsymbol{\Theta})$ with CDF $F_G(g;\boldsymbol{\Theta})$, the threshold η that yields the Top$_k$ vector with average target compression ratio $\delta \triangleq k/d$ can be derived as*

$$\eta(\delta) = F^{-1}_{|G|}(1-\delta;\widehat{\boldsymbol{\Theta}}) \tag{4}$$

$$= F^{-1}_{G}\left(1-\frac{\delta}{2};\widehat{\boldsymbol{\Theta}}\right), \tag{5}$$

where $\widehat{\boldsymbol{\Theta}}$ is the estimated parameters for the gradient distribution, $F_{|G|}(g;\widehat{\boldsymbol{\Theta}})$ is the CDF of the absolute gradient , $F^{-1}_{|G|}(p;\widehat{\boldsymbol{\Theta}}) \triangleq \left\{g \in \mathbb{R}^+ : F_{|G|}(g;\widehat{\boldsymbol{\Theta}}) = p\right\}$ is the inverse CDF of the absolute gradient at probability p, and $F^{-1}_{G}(p;\widehat{\boldsymbol{\Theta}})$ is the inverse CDF of the gradient, also known as quantile function or percent-point function (PPF).

Proof. From Property 2, the gradients can be modeled as r.v.s distributed according to a SID with CDF $F_G(g)$. Next, we would like to drive a threshold η such that on average the absolute values of k elements out of d are larger than η. The problem can be seen as a binomial random process, where the number of trials is d, the success probability is the probability that the absolute of the gradient is larger than η, i.e., $p \triangleq \mathbb{P}\{|G| \geq \eta\}$, and the average number of successes (exceedances) is k. In this process, the number of exceedances is a binomial r.v. with d number of trials and p probability of success (Papoulis & Pillai, 2002). The mean of the number of exceedances is dp. In order to have, on average, k elements out of d, we should have $\mathbb{P}\{|G| \geq \eta\} = \delta$. Hence, the threshold η is the $100(1-\delta)$ percentile of the distribution of absolute gradients as in (4). From the symmetry of the gradient distribution around zero, we have $\mathbb{P}\{|G| \geq \eta\} = 2\,\mathbb{P}\{G \leq -\eta\}$. Therefore, from (4), we get $\eta = -F_G^{-1}\left(\delta/2; \widehat{\mathbf{\Theta}}\right) = F_G^{-1}\left(1-\delta/2; \widehat{\mathbf{\Theta}}\right)$. $\square$

In the following, we report the threshold calculation for gradients modeled by double exponential distribution. The corresponding analysis for double gamma and GP is presented in Appendix B.3.

Corollary 1.1. *For double exponentially distributed gradients with scale parameter β and location zero (symmetric around zero), i.e., $G\sim \text{Laplace}(\beta)$, the threshold that achieves δ compression ratio can be computed as*

$$\eta = \hat{\beta} \log\left(\frac{1}{\delta}\right), \qquad \hat{\beta} \triangleq \frac{1}{d}\sum_{j=1}^{d} |g_j|, \tag{6}$$

where $\hat{\beta}$ is the maximum likelihood estimate (MLE) of the scale parameter.

Proof. For $G\sim \text{Laplace}(\beta)$, the gradient absolute is modeled as exponential distribution with scale β, $|G| \sim \text{Exp}(\beta)$ (Evans et al., 1994). From the inverse CDF of exponential distribution at probability p, i.e., $F_{|G|}^{-1} = -\beta\log(1-p)$, the MLE of β (Evans et al., 1994), and (4), the threshold in (6) follows. $\square$

Gradient compression through thresholding: After computing the threshold, the compressed gradient vector is found as $\widehat{g}_j = \mathbb{C}_\eta\{g_j\} \triangleq g_j\,\mathbb{I}_{\{|g_i|\geq\eta\}}$, for each $j \in \{1, 2, \cdots, d\}$, where the vector $\widehat{\mathbf{g}} \in \mathbb{R}^d$ is the compressed gradient vector, $\mathbb{I}_{\{\text{condition}\}}$ is an indicator function that equals one when the condition is satisfied and zero otherwise. In the following, we denote by $\bar{\mathbf{g}}$ and $\hat{k}$ the vector that contains only the exceedance non-zero gradients and their number, respectively.[5]

Possible issues in far tail fitting: The target compression ratio δ can be as low as 10^{-4}. Therefore, in order to accurately estimate the threshold, the fitted distribution should tightly resemble the gradient distribution at the tail. This is quite challenging because the estimation of the parameters tends to account more for the majority of data at the expense of the tail. Hence, the threshold obtained from single-stage fitting is accurate up to some moderate compression ratios. For lower compression ratios, the threshold tends to underestimate/overestimate the target δ. Hence, a more accurate tail fitting method is required to reduce the bias induced by the majority of non-significant gradients, as we show next.

2.4 Multi-Stage Threshold Estimator

We propose a multi-stage fitting approach to overcome the far tail estimation problem. For convenience, we start with the two-stage approach. First, the gradients are fitted with one of the three SIDs and compressed using the proposed procedure in Section 2.3 with a threshold η_1 computed to yield an initial compression ratio $\delta_1 \triangleq k_1/d > \delta$. Then, the vector of the exceedance gradients, $\bar{\mathbf{g}}$, is used to fit another distribution, defined precisely below. Then, another threshold η_2 is computed to achieve a compression ratio $\delta_2 \triangleq k_2/k_1$ with respect to the exceedance gradients. The second compression ratio is chosen such that the overall compression ratio of the original gradient vector is the target ratio δ, i.e., $\delta_2 = \delta/\delta_1$. Then, the estimated threshold from the last stage is applied to compress the original gradient vector. This procedure can be extended to multi-stages such that $\delta = \prod_{m=1}^{M} \delta_m$, where M is the number of stages.

The remaining question is whether the exceedance (known also as PoT) gradients have the same distribution as the original gradients before the compression. The extreme value theory in statistics can provide an answer for this question (Coles, 2001; Kotz & Nadarajah, 2000; Leadbetter, 1991; Smith, 1984). Let $\hat{k}_m$ be the number of exceedance gradients after the m^{th} thresholding stage. Then, if we apply a threshold operator on a sequence of r.v.s, $|G_1|, |G_2|, \cdots, |G_{\hat{k}_{m-1}}|$, the distribution of the exceedance r.v.s, $|\bar{G}_1|, |\bar{G}_2|, \cdots, |\bar{G}_{\hat{k}_m}|$, can be approximated by a GP distribution for large enough threshold and vector dimension, irrespective of the original distribution of the gradients.

Next, we exploit the extreme value theory to compute the threshold for the multi-stage approach.

Lemma 2. *Considering that for the m^{th} thresholding stage with $m \geq 2$, the absolute of the exceedance gradients, $|\bar{G}|_m$, can be modeled as $\text{GP}(\alpha_m, \beta_m, a_m)$, where $-1/2 < \alpha_m < 1/2$, β_m, and $a_m = \eta_{m-1}$ are the shape, scale, and location parameters. The threshold that achieves a compression ratio δ_m is obtained as*

$$\eta_m = \frac{\hat{\beta}_m}{\hat{\alpha}_m}\left(e^{-\hat{\alpha}_m \log(\delta_m)} - 1\right) + \eta_{m-1}, \tag{7}$$

$$\hat{\alpha}_m \triangleq \frac{1}{2}\left(1 - \frac{\bar{\mu}^2}{\bar{\sigma}^2}\right), \tag{8}$$

$$\hat{\beta}_m \triangleq \frac{1}{2}\,\bar{\mu}\left(\frac{\bar{\mu}^2}{\bar{\sigma}^2} + 1\right), \tag{9}$$

where η_{m-1} is the threshold computed at the previous stage and $\bar{\mu}$ and $\bar{\sigma}^2$ are the sample mean and variance of $|\bar{\mathbf{g}}_m| -$

[5]Note that the compressed vector $\widehat{g}_j$ coincides with the Top_k sparsified gradient with $k = \hat{k}$, i.e., $\mathbb{C}_\eta\{g_j\} = \mathbb{T}_{\hat{k}}\{g_j\}$.

η_{m-1}, respectively. For the proof of Lemma 2, please refer to Appendix B.3.3.

Corollary 2.1. *If the absolute of the gradients is modeled as exponentially distributed r.v.s, $|G_m| \sim \text{Exp}(\beta_m)$, the distribution of the exceedance gradients over the threshold η_{m-1}, after proper shifting, is still exponentially distributed, i.e., $|\bar{G}_m| - \eta_{m-1} \sim \text{Exp}(\beta_m)$. The new stage threshold is*

$$\eta_m = \hat{\beta}_m \log\left(\frac{1}{\delta_m}\right) + \eta_{m-1}, \tag{10}$$

$$\hat{\beta}_m \triangleq \frac{1}{\hat{k}_{m-1}} \sum_{j=1}^{\hat{k}_{m-1}} |\bar{g}_j| - \eta_{m-1}, \tag{11}$$

where $\bar{g}_j$ is the j^{th} element of the vector $\bar{\mathbf{g}}_m$. The proof is provided in Appendix B.3.4.

In the proposed scheme, we exploit Corollary 2.1 such that when the absolute of the gradients is fitted by an exponential distribution in the first stage, the latter stages for the exceedance gradients are also fitted by exponential distributions, i.e., multi-stage exponential. On the other hand, for gamma-fitted absolute gradients in the first stages, the latter stages are fitted by a GP distribution, from Lemma 2, i.e., gamma-GP. Finally, for GP distributed absolute gradients in the first stage, the GP is still used for the PoT data, from Lemma 2, i.e., multi-stage GP.

3 SIDCo Algorithm

SIDCo leverages SIDs to obtain a threshold via the multi-stage threshold estimator described in Section 2.4. We select the number of stages, M, via an adaptive algorithm such that the estimation error, averaged over Q iterations, is bounded below a predefined error tolerance, i.e,

$$\left|\hat{\delta} - \delta\right| \leq \epsilon\, \delta, \qquad 0 \leq \epsilon < 1\,. \tag{12}$$

First, we describe the algorithm that *SIDCo* follows to perform the gradient compression. The full pseudo-code is shown in Algorithm 1 of the Appendix. The algorithm in each iteration, takes as input the gradient vector and produces a compressed vector. The vector is sparsified through the multi-stage fitting strategy described in Section 2.4. In each stage, the function *Thresh_Estimation* uses the chosen SID to obtain a threshold. The algorithm dynamically adapts the number of stages M by monitoring the quality of its estimated selection of elements and adjusting M using function *Adapt_Stages*.

The algorithm starts by calling the *sparsify* function which takes the gradient and target ratio as the parameters. Then, the algorithm applies a multi-stage estimation loop of M iterations. In each iteration, the vector is partially sparsified with the previously estimated threshold obtained from the previous stage $m-1$. Then, given the ratio δ_m at loop step m, the chosen SID distribution fitting is invoked via the function *Thresh_Estimation* to obtain a new threshold. At the last stage (i.e., step M of the loop), the resulting estimation threshold should approximate the threshold that would obtain the target ratio δ of the input vector. Then, the estimated threshold is used to sparsify the full gradient vector and obtain the values and their corresponding indices. For each invocation of the algorithm in each training iteration, the algorithm maintains statistics like the average ratio of the quality of its estimations over the past training steps Q. Then, at the end of every Q training steps, the algorithm invokes *Adapt_Stages* which adjusts the current number of stages M based on user-defined allowable error bounds of the estimation (i.e., ϵ_H and ϵ_L). After the adjustment, the next algorithm invocation will use the new number of stages M. The number of stages is adjusted only if the obtained ratio is not within the error bounds.

3.1 Convergence Analysis

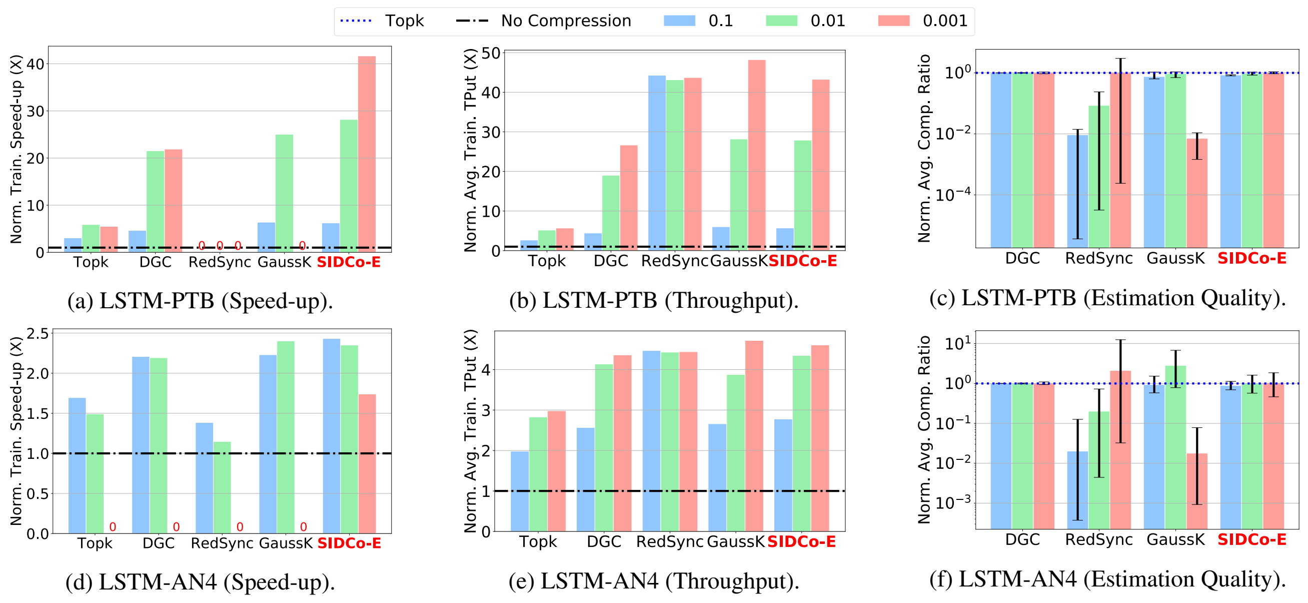
In the following, we present the convergence analysis of *SIDCo*.

Lemma 3. *Let $\hat{\delta}$ be the average achieved compression ratio of the proposed scheme with bounded discrepancy with respect to the target δ with error tolerance ϵ as in (12) which is assured by Algorithm 1 in the Appendix. Also, let i be the current training iteration, then the convergence rate of the proposed scheme coincides with that of the SGD if*

$$i > \mathbb{O}\left(\frac{1}{\delta^2\,(1-\epsilon)^2}\right). \tag{13}$$

Proof. The convergence of the proposed scheme would mainly follow the existing convergence analysis of Top$_k$ (Aji & Heafield, 2017; Alistarh et al., 2018; Stich et al., 2018), because *SIDCo* is designed to estimate a threshold for obtaining the top k elements. In contrast to Top$_k$, the number of non-zero elements in the proposed scheme is a binomial r.v., $\hat{K}$, and not a constant. Second, the expected value of the estimated number of non-zero elements, $\hat{k} \triangleq \mathbb{E}\{\hat{K}\}$, may not coincide with the target k, due to a possible mismatch between the assumed SID of the stochastic gradients and their original distribution. The complete proof is detailed in Appendix C. □

The proper selection of the distribution as a SID permits the actual compression rate to approach the designed compression ratio with small ϵ. This can be seen from the extensive numerical results in plots showing the estimation quality of Figure 1c, Figure 5b, Figure 6c, and Figure 9. One can notice that on average $\hat{k}/k \approx 1$, hence it resembles Top$_k$. For some rare extreme cases, we have $\hat{\delta} \geq 0.8\,\delta$ (i.e., $\epsilon = 20\%$), meaning that we need at most about 50% more iterations than Top$_k$ to reach the rate of SGD.



(a) LSTM-PTB (Speed-up). (b) LSTM-PTB (Throughput). (c) LSTM-PTB (Estimation Quality).

(d) LSTM-AN4 (Speed-up). (e) LSTM-AN4 (Throughput). (f) LSTM-AN4 (Estimation Quality).

Figure 3. Performance of training RNN-LSTM on PTB [(a),(b),(c)] and AN4 [(d),(e),(f)] datasets.

4 Experimental Evaluation

This evaluation answers the following questions:
- What benefits, in terms of training speed-up and model quality, does *SIDCo* provide compared to state-of-the-art approaches (*gains in training time to accuracy*)?
- Are the improvements of *SIDCo* only due to its faster training over other schemes (*training throughput gains*)?
- How accurate is the the threshold estimation of *SIDCo* compared to the state-of-the-art (*estimation quality*)?

In the following, we describe the main results, and present more experimental results and scenarios in Appendix E.

4.1 Experimental Settings

Unless otherwise mentioned, the default settings of the experiments are as follows.
Environment: We perform our experiments on 8 server machines equipped with dual 2.6 GHz 16-core Intel Xeon Silver 4112 CPU, 512GB of RAM, and 10 Gbps NICs. Each machine has an NVIDIA V100 GPU with 16GB of memory. The servers run Ubuntu 18.04, Linux kernel 4.15.0. We use PyTorch 1.1.0 with CUDA 10.2 as the ML toolkit. We use Horovod 0.16.4 configured with OpenMPI 4.0.0 for collective communication.

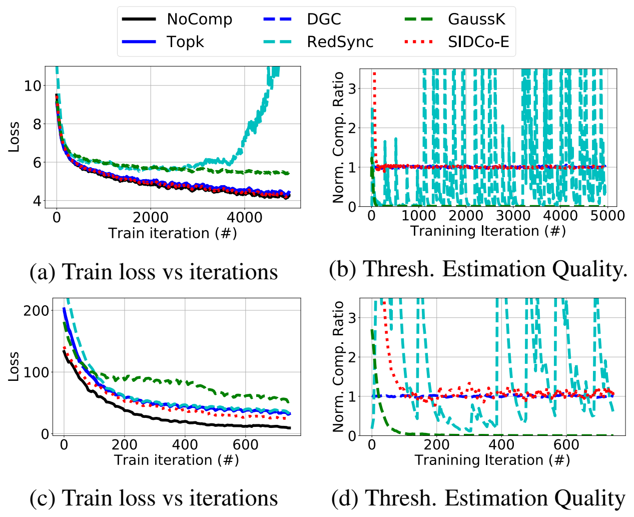
Benchmarks and hyper-parameters: The benchmarks and hyper-parameters are listed in Table 1. We use both CNN and RNN models for image classification and language modeling tasks, respectively. We use compression ratios (δ) of 0.1 (10%), 0.01 (1%) and 0.001 (0.1%) to span a wide range of the trade-off between compression and accuracy similar to prior work (Aji & Heafield, 2017; Alistarh et al., 2018; Lin et al., 2018). Further details of the environment, tasks and settings of the experiments are given in Appendix D.

Compressors: We compare *SIDCo* with Top$_k$, DGC, RedSync and GaussianKSGD. The EC mechanism is employed to further enhance the convergence of SGD with compressed gradients (Karimireddy et al., 2019; Lin et al., 2018). For *SIDCo*, we set $\delta_1 = 0.25$, $\epsilon = 20\%$, and $Q = 5$ iterations to adapt the stages as in Algorithm 1. For conciseness, we present the performance of *SIDCo* with double exponential fitting (shown in the figures as SIDCo-E).[6]

Metrics: We quantify the performance of a given scheme (i.e., *SIDCo*, Top-k, DGC, RedSync or GaussianKSGD) via the following metrics:
- *Normalized Training Speed-up:* We evaluate the model quality at iteration T (the end of training) and divide it by the time taken to complete T iterations. We normalize this quantity by the same measurement calculated for the baseline case. This is the normalized training speed-up relative to the baseline;
- *Normalized Average Training Throughput:* is the average throughput normalized by the baseline's throughput which illustrates the speed-up from compression irrespective of its impact on model quality;
- *Estimation Quality:* is the compression ratio ($\hat{k}/d$) averaged over the training divided by the target ratio ($\delta = k/d$) along with the 90% confidence interval as error-bars.

Next, we present the results for the benchmarks in Table 1.

[6]The results for double GP (SIDCo-GP) and double gamma (SIDCo-P), presented in Appendix F, are quite similar.



(a) Train loss vs iterations (b) Thresh. Estimation Quality. (c) Train loss vs iterations (d) Thresh. Estimation Quality

Figure 4. The training performance for the LSTM model on PTB and AN4 datasets with compression ratio of 0.001.

4.2 Recurrent Neural Networks (RNNs)

RNN-LSTM on PTB: This benchmark has the highest communication overhead (Table 1). In Figure 3a, *SIDCo* shows significant speed-up over no-compression by $\approx 41.7\times$ and improves over Top$_k$ and DGC by up to $\approx 7.6\times$ and $\approx 1.9\times$, respectively. At high compression ratio of 0.001, both RedSync and GaussianKSGD compression methods do not converge to the target loss and test perplexity (Figure 4a) and therefore they attain zero speed-ups. Figure 3b shows that threshold estimation schemes including *SIDCo* have the highest training throughput. However, in Figure 3c, DGC and *SIDCo* are the only methods that accurately estimate the target ratio with high confidence. However, for GaussianKSGD at ratio of 0.001 and RedSync at ratios of 0.01 and 0.001, the number of selected elements is two orders-of-magnitude lower than the target. Moreover, over the training process, the estimation quality of RedSync has high variance, harming convergence. Figure 4b shows, at target ratio of 0.001, RedSync causes significant fluctuation in compression ratio and training does not converge. GaussianKSGD results in very low compression ratio which is close to 0 and far from the target leading to significantly higher loss (and test perplexity) values compared to the target values.

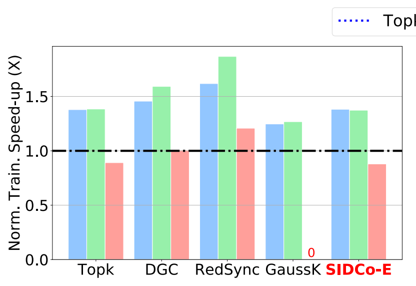
RNN-LSTM on AN4: Figure 3d shows that *SIDCo* achieves higher gains compared to other compressors by up to $\approx 2.1\times$ for ratios of 0.1 and 0.01. Notability, at ratio of 0.001, only *SIDCo* achieved the target character error rate (CER). Thus, we ran other compressors for 250 epochs to achieve the target CER (instead of the default 150), except for GaussianKSGD, which does not converge. The gains of *SIDCo* over the other compressors are increased by up to $\approx 4\times$. The reason could be that the model is more sensitive to compression (esp., in the initial training phase). *SIDCo* starts as single-stage before performing stage adaptations, leading to a slight over-estimation of k and so more gradient elements are sent during training start-up. Throughput-wise, Figure 3e shows that threshold-estimation methods including *SIDCo* enjoy higher training throughput, explaining the gains over the baseline. Similar to LSTM-PTB results, Figure 3f shows that on average, with low variance, *SIDCo* closely matches the estimated ratios of DGC while other estimation methods have poor estimation quality. Similar to PTB, Figure 4d shows, at target ratio of 0.001, RedSync causes significant fluctuation in compression ratio and GaussianKSGD results in very low compression ratio (close to 0) which is far from the target. This leads both methods to achieve significantly higher loss (or test perplexity) values compared to the target loss (or test perplexity) values.

4.3 Convolutional Neural Networks (CNNs)

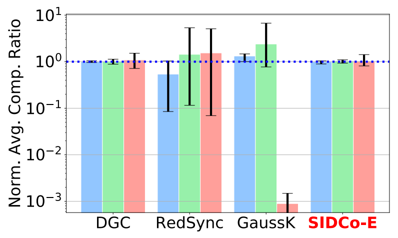
ResNet20 and VGG16 on CIFAR-10: Figure 5a shows that, for ResNet20, all compressors achieve somewhat comparable and modest speed-ups over the no-compression baseline (except at ratio of 0.001, where accuracy is degraded and hence the lower speed-up than the baseline). This is not surprising because ResNet20 is not network-bound. However, for the larger VGG16 model, Figure 5c shows that *SIDCo* achieves significant speed-ups over no-compression, Top$_k$ and DGC by up to $\approx 5\times$, $1.5\times$, and $1.2\times$, respectively. Figure 5b shows that, unlike other estimation schemes, *SIDCo* can accurately achieve the target ratio.

ResNet50 and VGG19 on ImageNet: In these experiments, we set a time-limit of 5 hours per run to reduce our costs. For calculating the speed-up, we compare the top-1 accuracy achieved by different methods at the end of training. First, for ResNet50 benchmark, we use compression ratios of 0.1, 0.01, and 0.001. Figure 6a shows that *SIDCo* achieves the highest accuracy that is higher than the baseline, Top$_k$ and DGC by ≈ 15, 3, and 2 accuracy points, i.e., normalized accuracy gains of $\approx 40\%$, 5%, and 4%, respectively. Figure 6b shows that *SIDCo* attains the highest throughput among all methods (except for RedSync at 0.1 compression). Figure 6c shows that, unlike GaussianKSGD and RedSync, which both result in estimation quality far from the target with high variance, *SIDCo* estimates the threshold with very high quality for all ratios. Similar trends are observed for the VGG19 benchmark where we use compression ratio of 0.001. As shown in Figures 6d to 6f, *SIDCo* estimates the threshold with high quality, and achieves the highest top-1 accuracy and training throughput among all methods. The accuracy gains compared to the baseline, Top$_k$ and DGC are $\approx 34\times$, $2.9\times$, and $1.13\times$, respectively.

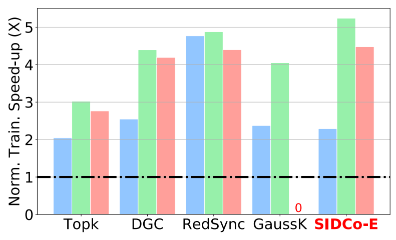
Takeaways: Our approach is simple in nature, which is intentional, to make it applicable in practice. Nonetheless,



(a) ResNet20-CIFAR-10 (Speedup).

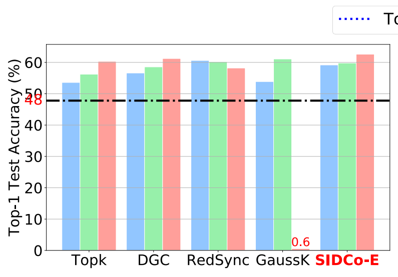


(b) ResNet20-CIFAR-10 (Est. Quality).

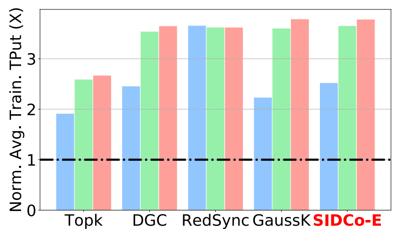


(c) VGG16-CIFAR-10 (Speedup)

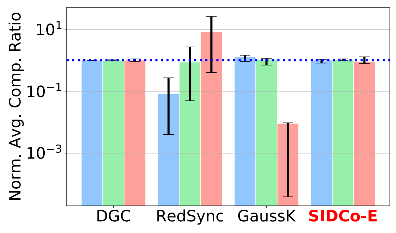
Figure 5. The training performance for ResNet20 [(a),(b)] and VGG16 [(c)] on CIFAR-10 dataset.



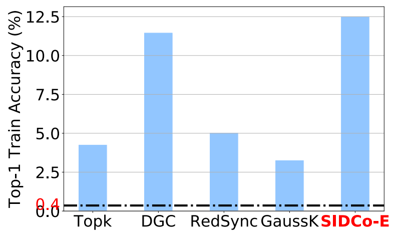
(a) ResNet50-ImageNet (Accuracy)



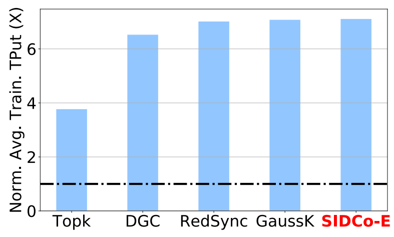
(b) ResNet50-ImageNet (Throughput)



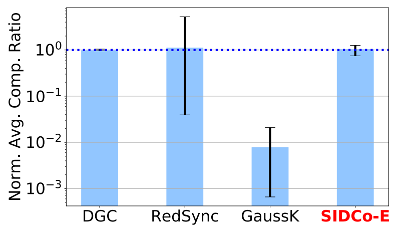
(c) ResNet50-ImageNet (Est. Quality)



(d) VGG19-ImageNet (Accuracy)



(e) VGG19-ImageNet (Throughput)



(f) VGG19-ImageNet (Est. Quality)

Figure 6. The training performance for ResNet50 [(a), (b), (c)] and VGG19 [(d), (e), (f)] on ImageNet dataset.

our work goes beyond existing works that estimate a threshold for Top$_k$ sparsification. These works either did not leverage the statistical property of the gradients (DGC) or assumed Gaussian distribution without a thorough study of the gradient (e.g., RedSync, GaussianKSGD). On a GPU, *SIDCo* improves over DGC by at least 2×, and the speedups are significantly larger on the CPU as shown in Figure 1b and Appendix E.2. As a threshold estimation method, SIDCo does not only benefit from the throughput gains of threshold methods but also from the high quality of its threshold estimation. The results in Figures 4 and 9 indicate that existing estimation methods (e.g., RedSync and GaussianKSGD) fail to achieve consistent threshold estimation behavior even though they may provide throughput gains. Their throughput gains, in many cases, are due to severe under-estimation of the target ratio, which results in lower volumes of data sent compared to other compressors.

5 Conclusion

We solved a practical problem in distributed deep learning. We showed that the performance of compressors other than threshold-based ones has high computational costs whereas existing threshold-estimation methods fail to achieve their target. To address these issues, we proposed *SIDCo* , a multi-stage threshold-based compressor through imposing a sparsity prior on the gradients. We evaluated *SIDCo* and compared it with popular compressors using common benchmarks involving RNN and CNN architectures. SIDCo, unlike existing threshold estimation methods, can efficiently approximate the target threshold and results in significant gains of up to $\approx 41.7\times$, $7.5\times$, and $1.9\times$ over no-compression baseline, Top$_k$ and DGC compression methods, respectively. Also, we expect further gains for large and communication-bounded models. In the future, we will explore ways to estimate a threshold for which compression satisfies other quality targets.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, 2016.

Abdelmoniem, A. M. and Canini, M. DC2: Delay-aware Compression Control for Distributed Machine Learning. In *INFOCOM*, 2021.

Abramowitz, M. and Stegun, I. A. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*, volume 55. US Government printing office, fourth edition, 1965.

Aji, A. F. and Heafield, K. Sparse Communication for Distributed Gradient Descent. In *EMNLP*, 2017.

Alistarh, D., Hoefler, T., Johansson, M., Konstantinov, N., Khirirat, S., and Renggli, C. The Convergence of Sparsified Gradient Methods. In *NeurIPS*, 2018.

AN4. CMU Census Database, 1991. http://www.speech.cs.cmu.edu/databases/an4/index.html.

Armagan, A., Dunson, D. B., and Lee, J. Generalized double Pareto shrinkage. *Statistica Sinica*, 23(1), 2013.

Babacan, S. D., Molina, R., and Katsaggelos, A. K. Bayesian Compressive Sensing Using Laplace Priors. *IEEE Transactions on Image Processing*, 19(1), 2010.

Baraniuk, R., Davenport, M. A., Duarte, M. F., and Hegde, C. An Introduction to Compressive Sensing, 2011. https://legacy.cnx.org/content/col11133/1.5/.

Beidi, C., Medini, T., Farwell, J., Gobriel, S., Tai, C., and Shrivastava, A. Slide : In defense of smart algorithms over hardware acceleration for large scale deep learning systems. In *MLSys*, 2020.

Bond, S. A. A review of asymmetric conditional density functions in autoregressive conditional heteroscedasticity models. In Knight, J. and Satchell, S. (eds.), *Return Distributions in Finance*, Quantitative Finance, chapter 2. Butterworth-Heinemann, Oxford, 2001.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. *arXiv 2005.14165*, 2020.

Coles, S. *An Introduction to Statistical Modeling of Extreme Values*. Springer London, 2001.

Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D. D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., Heinecke, A., Dubey, P., Corbal, J., Shustrov, N., Dubtsov, R., Fomenko, E., and Pirogov, V. O. Mixed precision training of convolutional neural networks using integer operations. In *ICLR*, 2018.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large Scale Distributed Deep Networks. In *NeurIPS*, 2012.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Li, F. F. ImageNet: a Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

DeVore, R. A. Nonlinear approximation. *Acta Numerica*, 7, 1998.

Dieuleveut, A. and Patel, K. K. Communication Trade-offs for Local-SGD with Large Step Size. In *NeurIPS*, 2019.

Dryden, N., Jacobs, S. A., Moon, T., and Van Essen, B. Communication Quantization for Data-Parallel Training of Deep Neural Networks. In *Workshop on ML in HPC (MLHPC)*, 2016.

Dutta, A., Bergou, E. H., Abdelmoniem, A. M., Ho, C.-Y., Sahu, A. N., Canini, M., and Kalnis, P. On the Discrepancy between the Theoretical Analysis and Practical Implementations of Compressed Communication for Distributed Deep Learning. In *AAAI*, 2020.

Elzanaty, A., Giorgetti, A., and Chiani, M. Limits on Sparse Data Acquisition: RIC Analysis of Finite Gaussian Matrices. *IEEE Transactions on Information Theory*, 65(3), 2019a.

Elzanaty, A., Giorgetti, A., and Chiani, M. Lossy Compression of Noisy Sparse Sources Based on Syndrome Encoding. *IEEE Transactions on Communications*, 67(10), 2019b.

Evans, M., Hastings, N., and Peacock, B. *Statistical distributions*. Wiley, New York, second edition, 1994.

Fang, J., Fu, H., Yang, G., and Hsieh, C.-J. RedSync: Reducing synchronization bandwidth for distributed deep learning training system. *Journal of Parallel and Distributed Computing*, 133, 2019.

Fu, F., Hu, Y., He, Y., Jiang, J., Shao, Y., Zhang, C., and Cui, B. Don't Waste Your Bits! Squeeze Activations and Gradients for Deep Neural Networks via TinyScript. In *ICML*, 2020.

Gajjala, R., Banchhor, S., Abdelmoniem, A. M., Dutta, A., Canini, M., and Kalnis, P. Huffman Coding Based Encoding Techniques for Fast Distributed Deep Learning. In *DistributedML*, 2020.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv 1706.02677*, 2017.

Gross, S. and Wilber, M. Training and investigating Residual Nets, 2016. http://torch.ch/blog/2016/02/04/resnets.html.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. Deep speech: Scaling up end-to-end speech recognition. *arXiv 1412.5567*, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *Proc. of CVPR*, pp. 770–778, 2015.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computing*, 9(8), 1997.

Hosking, J. and Wallis, J. Parameter and Quantile Estimation for the Generalized Pareto Distribution. *Technometrics*, 29(3), 1987.

Jiang, J., Fu, F., Yang, T., and Cui, B. SketchML: Accelerating Distributed Machine Learning with Data Sketches. In *SIGMOD*, 2018.

Karimireddy, S. P., Rebjock, Q., Stich, S., and Jaggi, M. Error Feedback Fixes Sign SGD and other Gradient Compression Schemes. In *ICML*, 2019.

Kotz, S. and Nadarajah, S. *Extreme Value Distributions: Theory and Applications*. World Scientific, 2000.

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

Kurth, T., Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E., Mahesh, A., Matheson, M., Deslippe, J., Fatica, M., Prabhat, P., and Houston, M. Exascale Deep Learning for Climate Analytics. In *SC*, 2018.

Leadbetter, M. R. On a basis for 'Peaks over Threshold' modeling. *Statistics & Probability Letters*, 12(4), 1991.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *ICLR*, 2018.

Mallat, S. *A Wavelet Tour of Signal Processing: The Sparse Way*. Academic Press, 2009.

Marcus, M., Santorini, B., Marcinkiewicz, M., and Taylor, A. Treebank-3, 1999. https://catalog.ldc.upenn.edu/LDC99T42.

Minka, T. P. Estimating a Gamma distribution. Technical report, Microsoft Research, 2002.

Monga, V., Mousavi, H. S., and Srinivas, U. Sparsity Constrained Estimation in Image Processing and Computer Vision. In *Handbook of Convex Optimization Methods in Imaging Science*, pp. 177–206. Springer International Publishing, 2018.

Narang, S., Diamos, G., Elsen, E., Micikevicius, P., Alben, J., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *ICLR*, 2018.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *SOSP*, 2019.

Olver, F. *Asymptotics and special functions*. CRC Press, 1997.

Papoulis, A. and Pillai, S. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 2002.

Peng, Y., Zhu, Y., Chen, Y., Bao, Y., Yi, B., Lan, C., Wu, C., and Guo, C. A Generic Communication Scheduler for Distributed DNN Training Acceleration. In *SOSP*, 2019.

pytorch.org. PyTorch. https://pytorch.org/.

Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv 1802.05799*, 2018.

Shanbhag, A., Pirk, H., and Madden, S. Efficient Top-K Query Processing on Massively Parallel Hardware. In *SIGMOD*, 2018.

Shi, S., Chu, X., Cheung, K. C., and See, S. Understanding Top-k Sparsification in Distributed Deep Learning. *arXiv 1911.08772*, 2019.

Shi, S., Zhou, X., Song, S., et al. Towards scalable distributed training of deep learning on public cloud clusters. *arXiv preprint arXiv:2010.10458*, 2020.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *ArXiv 1909.08053*, 2019.

Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015.

Smith, R. L. Threshold Methods for Sample Extremes. In *Statistical extremes and applications*. Springer, 1984.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. Sparsified SGD with Memory. In *NeurIPS*, 2018.

Vanhoucke, V., Senior, A., and Mao, M. Z. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop - NeurIPS*, 2011.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J. S. A Survey on Distributed Machine Learning. *ACM Computing Surveys*, 53(2), 2020.

Wangni, J., Wang, J., Liu, J., and Zhang, T. Gradient Sparsification for Communication-Efficient Distributed Optimization. In *NeurIPS*, 2018.

Wu, J., Huang, W., Huang, J., and Zhang, T. Error Compensated Quantized SGD and its Applications to Large-scale Distributed Optimization. In *ICML*, 2018.

Xu, H., Ho, C.-Y., Abdelmoniem, A. M., Dutta, A., Bergou, E. H., Karatsenidis, K., Canini, M., and Kalnis, P. Compressed communication for distributed deep learning: Survey and quantitative evaluation. Technical report, KAUST, 2020. http://hdl.handle.net/10754/662495.

Algorithm 1 *SIDCo* Algorithm

```
Input: g  the gradient vector to be compressed
Input: $\delta$  the target compression ratio
Input: $i$  the training iteration number
Input: $Q$: the frequency of invoking stages adaption
Input: ($\epsilon_H$,$\epsilon_L$): upper and lower bounds of estimation error for adapting the stages.
1  /* The discrepancy tolerance $\epsilon$ in (12) can be computed as $\epsilon = \max(\epsilon_H, \epsilon_L)$ */
2  Define $\hat{k}, \hat{k}_{avg}$: number of and average number of elements obtained from threshold estimation.
   Function Sparsify($\mathbf{g}, \delta$)
3      /* Multi-stage threshold estimation */
4      $\mathbf{temp} = copy(\mathbf{g})$
       $\eta_0 = 0$
       for each stage $m$ in (1,$M$) do
5          $\mathbf{temp} = \mathbf{temp} - \eta_{m-1}$
           $\eta_m$ = Thresh_Estimation($\mathbf{temp}, \delta_m, DIST$)
           $\widehat{I} = abs(\mathbf{temp}) > \eta_m$ - find the index of top absolute values larger than $\eta_m$.
           Use $\widehat{I}$ to filter temp which keeps the top values of temp and sets the others to zero.
6      if $i \mod Q == 0$ then
7          /* Call stages adaption function to adjust number of stages */
8          $M$=Adapt_Stages($M$, $\frac{\hat{k}_{avg}}{Q}$ )
           $\hat{k}_{avg} = 0$
9      else
10         $\hat{k}_{avg} = \hat{k}_{avg} + \hat{k}$
11     /* Threshold sparsification: find indices of top absolute values larger than
          the threshold $\eta_M$, then use them to filter the elements of the gradient g */
12     return $\mathbb{C}_{\hat{k}}(\mathbf{g}) = \mathbf{g}[abs(\mathbf{g}) \geq \eta_M]$
13 Function Thresh_Estimation($\mathbf{g}$, $\delta$, DIST)
14     if DIST is Exponential then
15         $\widehat{\mu}$ = mean(abs($\mathbf{g}$))
           return $-\widehat{\mu}\ \log(\delta)$
16     if DIST is GPareto then
17         $\widehat{\mu}, \widehat{\sigma}$ = mean_var(abs($\mathbf{g}$))
           $\hat{\alpha} = 0.5\left(1 - \frac{\widehat{\mu}^2}{\widehat{\sigma}^2}\right)$
           $\hat{\beta} = 0.5\,\widehat{\mu}\left(\frac{\widehat{\mu}^2}{\widehat{\sigma}} + 1\right)$
           return $\frac{\hat{\beta}}{\hat{\alpha}}\ (\exp(-\hat{\alpha}\log(\delta)) - 1)$
18     if DIST is Gamma then
19         $\widehat{\mu}$ = mean(abs($\mathbf{g}$))
           $s$ = log($\widehat{\mu}$) - mean(log(abs($\mathbf{g}$)))
           $\hat{\alpha} = \frac{3 - s + \sqrt{(s-3)^2 + 24\,s}}{12\,s}$
           $\hat{\beta} = \frac{\widehat{\mu}}{\hat{\alpha}}$
           return $-\hat{\beta}\,(\log(\delta) + \log(\text{gamma}(\hat{\alpha})))$
20 Function Adapt_Stages($M, \hat{k}_{avg}$)
21     /* Choose the number of stages */
22     if $\hat{k}_{avg} > k * (1 + \epsilon_H)$ then
23         $M = M - 1$
24     if $\hat{k}_{avg} < k * (1 - \epsilon_L)$ then
25         $M = M + 1$
26     return min(max($M$, 1),$M_{max}$)
```

A Distributed Synchronous SGD (DSSGD) with Sparsification

Here, we describe the Distributed Synchronous version of SGD optimization algorithm which is the main work-horse behind most of the distributed training (Aji & Heafield, 2017). Algorithm 2 presents the specifics of the algorithm with sparsification. The algorithm executes in parallel on each worker and starts with sampling a batch-sized sub-set from the full training dataset. Then, each worker performs a local pass including forward pass to obtain a loss value followed by a backward pass to calculate the gradient vector for updating the model parameters. At this point and before moving along with the model update, the workers have to synchronize by performing aggregation (or averaging) of their gradient vectors and use the aggregated gradient for the update. The gradient is sparsified using a chosen compressor (e.g., Top$_k$, DGC, *SIDCo*, .., etc) and target sparsification ratio. For example, to invoke *SIDCo* compressor, one would invoke function *S*parsify of Algorithm 1 which takes as input the gradient g and target sparsification ratio δ. Then, the aggregation can be either accomplished via means of a parameter server which has a global copy of the model parameters and receives the gradients from the workers and update its local model and then the workers can pull the up-to-date model parameters at any time (Dean et al., 2012). The other way is to perform aggregation in a peer-to-peer fashion via means of collective operation like All-Reduce or All-Gather which requires no extra parameters server for the aggregation (Sergeev & Balso, 2018). The peer-to-peer collective communication methods are widely adopted by most frameworks (pytorch.org; Sergeev & Balso, 2018) and known to scale well in practice (Goyal et al., 2017) and hence is adopted in this work.

A.1 Discussion on *SIDCo* Algorithm

We highlight a few technical aspects of *SIDCo* algorithm presented in Algorithm 1. **Scalability concerns:** The compression algorithm has no scalability issues since it executes locally and does not rely on inter-node communication. Also, the compressor only depends on the size of the gradient vector leading to the same compression time on all the workers regardless of the number of training workers that run in parallel.

Algorithm's dependence on training iteration: the gradient sparsity changes over iterations as shown in Figure 2 and Figure 8. The proposed algorithm leverages extreme-value theorem to handle sparsity variations by adapting the number of stages at each iteration. This enables adaptive fitting of the gradient at each iteration via the sparsity-inducing distribution enabling the estimation of an approximate threshold that obtains the top k elements of the gradient vector.

Algorithm 2 Sparsified Distributed Synchronous SGD

```
   /* Worker n */
27 /* Initialization */
   Input: $D$: Local Dataset
   Input: $B$: Minibatch size per node
   Input: $N$: The total number of workers
   Input: $\lambda$: The learning rate
   Input: $\delta$: The target sparsification ratio
   Input: $x$: Model parameters $x = (x[0], x[1], ..., x[d])$
28 /* loop till end of training */
29 for i = 0, 1, ... do
30     /* Calculate stochastic gradient */
31     $\mathbf{g}^n_{\{i\}} = 0$
       for i = 1, ..., B do
32         Sample data point $d$ from $D$
           Calculate $\nabla f(x; d)$
           $\mathbf{g}^n_{\{i\}} = \mathbf{g}^n_{\{i\}} + \frac{1}{B}\nabla f(x; d)$
33     /* Aggregate workers' gradients */
34     Collective-Comm: $G^n_{\{i\}} = \frac{1}{N}\sum_{n=1}^{N}$ Sparsify$(\mathbf{g}^n_{\{i\}}, \delta)$
       /* Update model parameters */
35     $\mathbf{x}^n_{\{i+1\}} = \mathbf{x}^n_{\{i\}} + \lambda\, G^n_{\{i\}}$
```

Sparsity and compressability of the gradients: our algorithm relies on a principled statistical approach, which makes it robust to various sparsity levels of the gradient given that the compressibility property holds. And if not, most sparsifiers would be equally ineffective. Moreover, the compressibility property is the reason why Top$_k$ is commonly used in the literature. Therefore, in this work, we seek an approximate fast threshold estimation method that exploits a common prior information of the gradients, while preserving the convergence guarantee of Top$_k$, albeit with different rate depending on the accuracy of the estimated threshold.

B Gradient Features and Distribution

B.1 Validation of Gradient Compressibility

The compressibility of the gradient vector allows efficient compression for the gradients through sparsification techniques, e.g., Top$_k$ and thresholding-based compression (Elzanaty et al., 2019b; Xu et al., 2020). Here, we empirically investigate the compressibility of the gradients according to Definition 1. In order to verify the vector compressibility, we consider the gradients generated while the training of ResNet20. The absolute of the gradients are sorted in descending order to obtain the vector $\tilde{\mathbf{g}}$ with $d = 269722$. In Figure 7a, the elements of the gradient vector $\tilde{\mathbf{g}}$, i.e., $\tilde{g}_j$, are reported vs their index, for three iterations in the beginning, middle, and end of the training.[7] As a benchmark, we report a power low decay example with decay exponent $p > 0.5$, i.e., $p = 0.7$. It can be noticed that the gradients follow a power-law decay with decay exponent $p > 0.7 > 0.5$; hence, they are compressible from (1).

In Figure 7b, we report the sparsification error for the best k approximation, e.g., the Top_k, as a function of k. We also report an example of the power decay model with decay exponent $p - 0.5 = 0.2$. We can see the best k approximation error decays faster than the benchmark. Hence, the vector can be considered compressible, according to (2).

We also validate this behavior for various models and datasets, not reported here for conciseness. Therefore, the gradient vectors can be considered compressible in the sense of Definition 1.

B.2 Validation of Gradient Distributions

In this part, we discuss the distribution of the gradients generated while training several neural networks. Since the gradient vectors are compressible, SIDs can approximate the gradient distribution. This feature, i.e. Property 2, is numerically validated as follows.

First, we consider the gradients from training ResNet-20 with SGD. The generated gradient vector at iteration i is compressed using Top_k with $\delta = 0.001$, and the distributed SGD is employed as described in Appendix A. We investigate two cases: i) memoryless compression, where the Error compensation (EC) mechanism is not applied, as shown in Figure 2 ii) memory-based compression, where an EC mechanism is deployed by adding the sparsification error from the previous iteration to the gradient vector before the Top_k compression, i.e., $\mathbf{g}_{\{i\}} = \mathbf{g}_{\{i\}} + \left[\mathbf{g}_{\{i-1\}} - \mathbb{T}_k \left\{\mathbf{g}_{\{i-1\}}\right\}\right]$. For both cases, we collect the uncompressed gradients from the master worker, as different workers have similar gradients in distributions. The gradient vectors are then normalized by their ℓ_2 norm to easily visualize and compare the evolution of the gradient distributions over various iterations. Then, the collected gradients are fitted by the three proposed SIDs, i.e., double exponential, double gamma, and double GP distributions. The parameters of the distribution are estimated as indicated in Corollary 1.1, Corollary 1.2, and Corollary 1.3.

For the training with EC mechanism in Figure 8, it becomes more challenging to fit the gradients, especially for larger iterations, as can be seen in Figure 8c. This behavior arises due to the addition of the sparsification error from the previous stage to the gradients, as the resulting distribution of the gradients changes. More precisely, the gradient distribution is the convolution between the PDF of the error from the last stage and PDF of the current gradient vector before the EC. Therefore, the distribution of the gradients significantly changes over the iterations. Therefore, single-stage fitting suffers more when EC mechanism is used, particularly for fitting the tail, as in Figure 8b and Figure 8d.

[7]Note that in Figure 7a, we focus only on the elements from 1 to 10^5, as for larger indices the amplitude of the vector elements are sufficiently small.

We also validate that the gradients generated from the other networks in Table 1 can be well approximated with r.v.s distributed according to one of the SIDs, which are not reported here for conciseness. In general, there is a slight variation in the fitting accuracy among the three SIDs for various networks and datasets, due to the nature of the gradients. For example, the double exponential distribution can not capture well the gradients with an empirical distribution that decays fast. In contrast, the double gamma and double GP distributions have an additional shape parameter that can approximate the behavior of sparser vectors with $\alpha < 1$. Nevertheless, the double-exponential behaves well when the distribution of the absolute of the gradients decays as fast as the exponential distribution.

B.3 Analysis of Double Gamma and Double Generalized Pareto Distributed Gradients

B.3.1 Threshold Calculation for Gamma Distributed Gradients

Corollary 1.2. *Considering that the gradients that can be well-fitted by double gamma distribution with shape parameter $\alpha \leq 1$, the absolute of the gradient is gamma distributed (Bond, 2001), i.e., $|G| \sim \text{gamma}(\alpha, \beta)$. The sparsifying threshold can be derived as*

$$\eta(\delta) = \hat{\beta}\, P^{-1}(\hat{\alpha}, 1-\delta) \tag{14}$$

$$\simeq -\hat{\beta}\,[\log(\delta) + \log(\Gamma(\hat{\alpha}))], \tag{15}$$

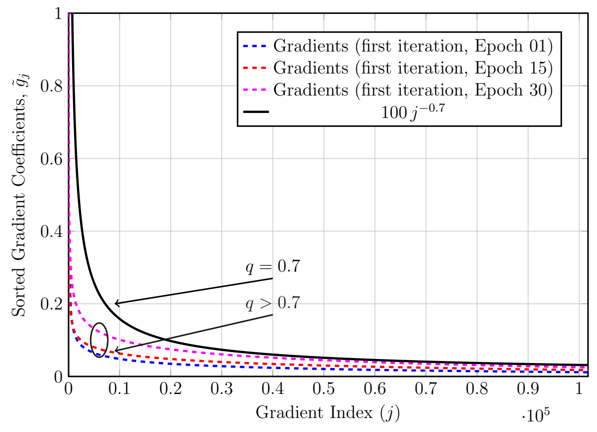
where $P(\alpha, x) \triangleq \frac{1}{\Gamma(\alpha)} \int_0^x t^{\alpha-1}\, \mathrm{e}^{-t}\, \mathrm{d}t$ is the regularized lower incomplete gamma function, and $P^{-1}(\alpha, p) \triangleq \{x : P(\alpha, x) = p\}$ is the inverse of the regularized lower incomplete gamma function (Abramowitz & Stegun, 1965),

$$\hat{\alpha} \triangleq \frac{3 - s + \sqrt{(s-3)^2 + 24\, s}}{12\, s}, \qquad \hat{\beta} \triangleq \frac{\widehat{\mu}}{\hat{\alpha}}, \tag{16}$$

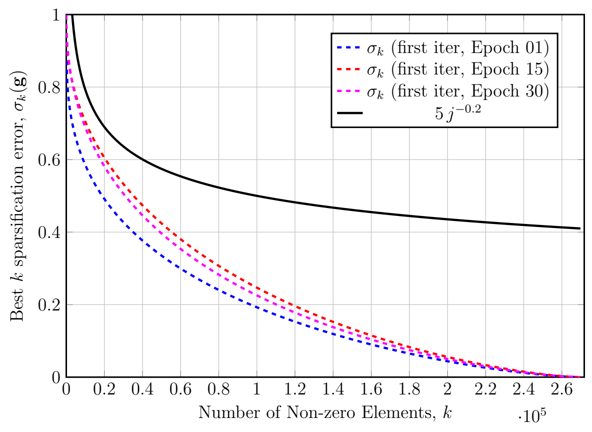
with $s \triangleq \log(\widehat{\mu}) - \widehat{\mu}_{\log}$, $\widehat{\mu}_{\log} \triangleq \frac{1}{d}\sum_{i=1}^{d} \log\left(|g|_i\right)$, and $\widehat{\mu}$ and $\widehat{\sigma}^2$ are the sample mean and variance for the absolute gradient vector $|\mathbf{g}|$, respectively.

Proof. The gradients are modeled by double gamma distribution with $\alpha \leq 1$, with PDF defined in (Bond, 2001) as

$$f_G(g; \alpha, \beta) = \frac{1}{2} \frac{|g|^{\alpha-1} e^{-|g|/\beta}}{\beta^{\alpha}\Gamma(\alpha)}, \quad \text{for} \quad -\infty < g < \infty. \tag{17}$$

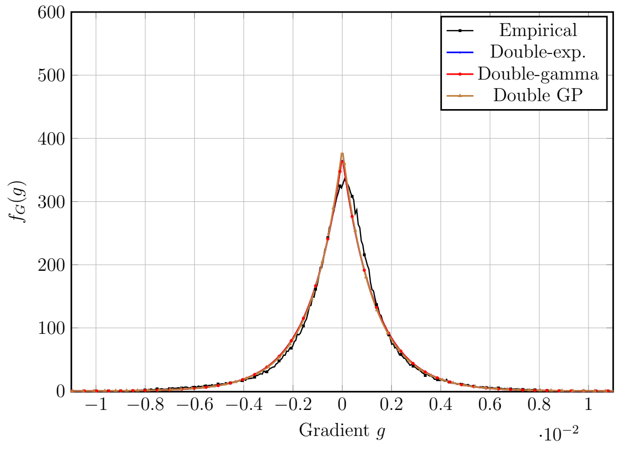


(a) The sorted magnitude of the gradients vs their indexes, and the fitted curve via power law in (1).

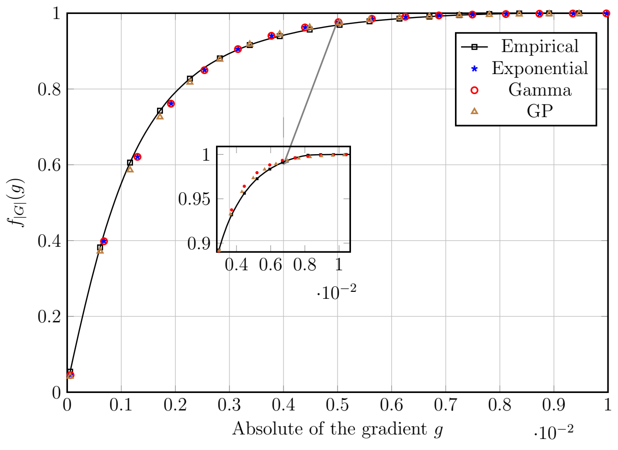


(b) The approximation error for the Top$_k$ vs the number of non-zero elements, k.

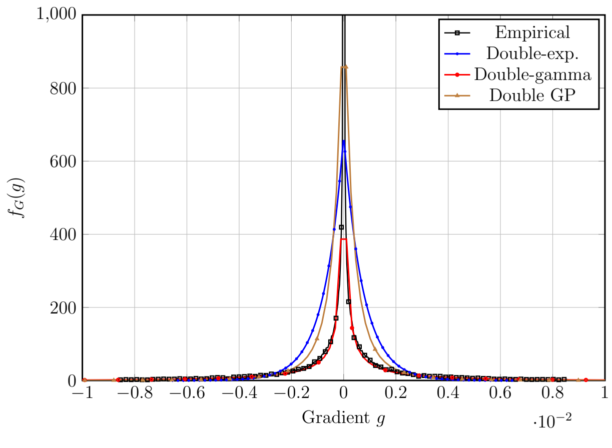
Figure 7. The compressibility property of the gradients



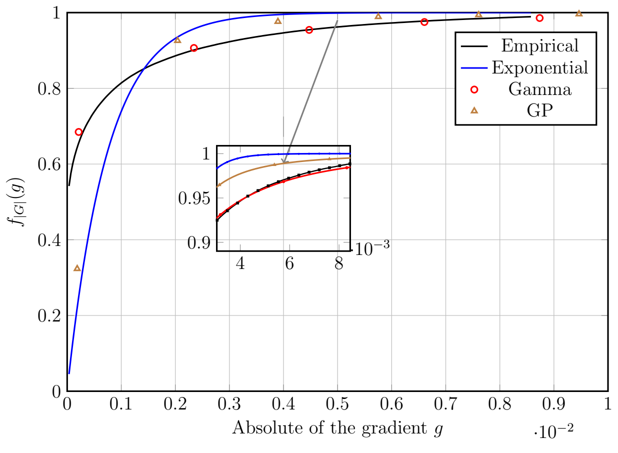
(a)



(b)



(c)



(d)

Figure 8. Gradient fitting using the three SIDs for the gradient vector along with the empirical distribution generated from training ResNet-20 on CIFAR10 dataset using Top$_k$ compressor **with EC mechanism**, for the 100th [(a) PDF, (b) CDF] and 10000th [(c) PDF, (d) CDF] iterations.

Hence, the absolute of the gradient is modeled as gamma distribution with PDF

$$f_{|G|}(g;\alpha,\beta) = \frac{g^{\alpha-1}e^{-g/\beta}}{\beta^{\alpha}\Gamma(\alpha)}, \quad \text{for} \quad 0 \le g < \infty. \tag{18}$$

The CDF of the gamma r.v. which can be written from (18) as

$$F_{|G|}(g;\alpha,\beta) = \int_0^g \frac{t^{\alpha-1}e^{-t/\beta}}{\beta^{\alpha}\Gamma(\alpha)}\mathrm{d}t \tag{19}$$

$$= \int_0^{g/\beta} \frac{z^{\alpha-1}e^{-z}}{\Gamma(\alpha)}\mathrm{d}z \triangleq P(\alpha, g/\beta), \tag{20}$$

where $P(\alpha, x)$ is the regularized lower incomplete gamma function (Abramowitz & Stegun, 1965). The threshold in (14) follows from the inverse of the CDF at $1-\delta$, as illustrated in (4) and by substituting the parameters of the gamma distribution with their estimates $\hat{\alpha}$ and $\hat{\beta}$. Nevertheless, calculating the threshold involves the inverse of incomplete gamma function which can be computationally heavy. In the following we provide a closed-form approximation for the threshold. First, we would like to find a closed-form approximation for the inverse lower incomplete function at $1-\delta$, i.e., $x \triangleq P^{-1}(\hat{\alpha}, 1-\delta)$. Starting from the bound on $P(\hat{\alpha}, x)$ in (Olver, 1997), we have

$$P(\hat{\alpha}, x) = 1-\delta \ge 1 - \frac{x^{\hat{\alpha}-1}\, e^{-x}}{\Gamma(\hat{\alpha})} \qquad \text{for } \hat{\alpha} \le 1, x > 0. \tag{21}$$

After some manipulations, we get

$$x \le -\log\left(\delta\Gamma(\hat{\alpha})\right) - (1-\hat{\alpha})\log(x), \quad \text{for } \hat{\alpha} \le 1, x > 0, \tag{22}$$

$$x \le -\log\left(\delta\right) - \log\left(\Gamma(\hat{\alpha})\right), \qquad \text{for } \hat{\alpha} \le 1, x \ge 1. \tag{23}$$

Finally, by substituting $P^{-1}(\hat{\alpha}, 1-\delta)$ with (23) in (14), we get

$$\eta \le -\hat{\beta}\,\left[\log(\delta) + \log(\Gamma(\hat{\alpha}))\right], \quad \text{for } \hat{\alpha} \le 1, x \ge 1 \tag{24}$$

with equality if $\hat{\alpha} = 1$. For $0 < x < 1$ or $\hat{\alpha} > 1$, the bound does not hold, however, it provides a good approximation for the threshold when $\hat{\alpha}$ is close to one.

For estimating the pentameters, let us start by the PDF of the gamma distribution, defined as

$$f_{|G|}(g;\alpha,\beta) = \frac{g^{\alpha-1}e^{-g/\beta}}{\beta^{\alpha}\Gamma(\alpha)} \qquad \text{for } x > 0, \quad \alpha, \beta > 0 \tag{25}$$

where α and β are the shape and scale parameters, respectively. The shape parameter can be estimated from the absolute gradient vector $|\mathbf{g}|$ using MLE as the solution of

$$\Psi(\alpha) - \log(\alpha) + \log(\widehat{\mu}) - \widehat{\mu}_{\log} = 0, \tag{26}$$

where $\Psi(x) \triangleq \frac{d\Gamma(x)}{dx}$ is the digamma function, $\widehat{\mu} \triangleq \frac{1}{d}\sum_{i=1}^{d}|g|_i$ is the sample mean, and $\widehat{\mu}_{\log} \triangleq \frac{1}{d}\sum_{i=1}^{d}\log\left(|g|_i\right)$ (Papoulis & Pillai, 2002). On the other hand, the scale parameter can be estimated as $\hat{\beta} = \widehat{\mu}/\alpha$. Nevertheless, the shape parameter estimation in (26) involves solving a non-linear equation with a special function. Hence, it increases the computational complexity for the scheme, leading to higher time overhead for the compression. In order to reduce the complexity, we propose to employ a simpler closed-form approximation for the shape parameter, i.e.,

$$\hat{\alpha} = \frac{3 - s + \sqrt{(s-3)^2 + 24\,s}}{12\,s}, \qquad \hat{\beta} = \frac{\widehat{\mu}}{\alpha}, \tag{27}$$

where $s \triangleq \log(\widehat{\mu}) - \widehat{\mu}_{\log}$ (Minka, 2002). $\square$

B.3.2 Threshold Calculation for Generalized Pareto Distributed Gradients

Corollary 1.3. *For gradients distributed as double generalized Pareto r.v.s, the absolute of the gradients is modeled as GP distributed r.v.s, $|G| \sim \mathrm{GP}(\alpha, \beta, a)$, where $0 < \alpha < 1/2$, β, $a = 0$ are the shape, scale, and location parameters. The sparsifying threshold that achieves a compression ratio δ is*

$$\eta = \frac{\hat{\beta}}{\hat{\alpha}}\left(e^{-\hat{\alpha}\log(\delta)} - 1\right), \tag{28}$$

where

$$\hat{\alpha} \triangleq \frac{1}{2}\left(1 - \frac{\hat{\mu}^2}{\hat{\sigma}^2}\right), \qquad \hat{\beta} \triangleq \frac{1}{2}\,\hat{\mu}\left(\frac{\hat{\mu}^2}{\hat{\sigma}^2} + 1\right), \tag{29}$$

with $\hat{\mu}$ and $\hat{\sigma}^2$ being the sample mean and variance for the absolute gradient vector, $|\mathbf{g}|$, respectively.

Proof. the gradients can be well-fitted by double GP distribution with PDF, indicated in (Armagan et al., 2013) as[8]

$$f_G(g) = \frac{1}{2\beta}\left(1 + \alpha\,\frac{|g|}{\beta}\right)^{-(\frac{1}{\alpha}+1)}, \qquad 0 < \alpha < \frac{1}{2}, -\infty < g < \infty \tag{30}$$

Hence, the absolute of the gradients can be modeled as GP distributed r.v.s with PDF

$$f_{|G|}(g) = \frac{1}{\beta}\left(1 + \alpha\,\frac{g}{\beta}\right)^{-(1/\alpha+1)}, \quad 0 < \alpha \le \frac{1}{2}, g \ge 0 \tag{31}$$

[8]The double GP distribution resembles the Laplacian distribution for $\alpha \to 0$. Similarly, the GP becomes exponential distribution for $\alpha = 0$.

and the corresponding CDF can be written from (Hosking & Wallis, 1987) as

$$F_{|G|}(g) = 1 - \left(1 + \alpha\,\frac{g}{\beta}\right)^{-1/\alpha}. \tag{32}$$

The inverse CDF can be written from (32) as

$$F^{-1}_{|G|}(p) = \frac{\beta}{\alpha}\left(e^{-\alpha\log(1-p)} - 1\right). \tag{33}$$

From (4) and (33) and by substituting the distribution parameters with their estimates, provided below, the threshold in (28) follows.

Unfortunately, there are no closed-form maximum likelihood (ML) estimators for the parameters of GP distributions. Hence, the ML estimates have to be computed through complex numerical optimization. Alternately, the parameters can be estimated in closed-from through the moment matching (MM) method under some conditions on the shape parameter (Hosking & Wallis, 1987). More precisely, for the considered range of the shape parameter, i.e., $-0.5 < \alpha < 0.5$, the first and second moments exit and they can be written as

$$\mu = \frac{\beta}{1+\alpha}, \qquad S^2 = \frac{\beta^2}{(1+\alpha)^2(1+2\,\alpha)}, \tag{34}$$

where μ and S^2 are the mean and mean square, respectively. Therefore, from (34) through the MM method, the parameters can e estimated as

$$\hat{\alpha} = \frac{1}{2}\left[1 - \frac{\hat{\mu}^2}{\hat{\sigma}^2}\right], \qquad \hat{\beta} = \frac{1}{2}\,\hat{\mu}\left[\frac{\hat{\mu}^2}{\hat{\sigma}^2} + 1\right], \tag{35}$$

where $\hat{\mu}$ and $\hat{\sigma}^2$ are the sample mean and variance for the absolute gradient vector, $|\mathbf{g}|$, respectively. $\square$

B.3.3 *Proof of Lemma 2*

The distribution of the PoT absolute gradients for the mth stage can be approximated as GP distribution from Theorem 4.1 in (Coles, 2001) with CDF

$$F_{|\bar{G}_m|}(g) = 1 - \left(1 + \alpha_m\,\frac{g - \eta_{m-1}}{\beta_m}\right)^{-1/\alpha_m}, \quad g \geq \eta_{m-1}, -1/2 < \alpha_m < 1/2, \tag{36}$$

where the first and second moments of the r.v. $|\bar{G}_m|$ are finite and the PDF is smooth for $-1/2 < \alpha_m < 1/2$ (Hosking & Wallis, 1987). The inverse CDF can be written from (36) as

$$F^{-1}_{|\bar{G}_m|}(p) = \frac{\beta_m}{\alpha_m}\left(e^{-\alpha_m\log(1-p)} - 1\right) + \eta_{m-1}. \tag{37}$$

The threshold in (7) follows from (4) and (37) and by substituting the distribution parameters with their estimates derived as from (34)

$$\hat{\alpha}_m = \frac{1}{2}\left[1 - \frac{\bar{\mu}^2}{\bar{\sigma}^2}\right], \qquad \hat{\beta}_m = \frac{1}{2}\,\bar{\mu}\left[\frac{\bar{\mu}^2}{\bar{\sigma}^2} + 1\right], \tag{38}$$

where the sample mean $\bar{\mu}$ and the variance $\bar{\sigma}^2$ are computed from absolute of the PoT gradients shifted by the threshold, i.e., $|\tilde{\mathbf{g}}_m| - \eta_{m-1}$.

B.3.4 *Proof of Corollary 2.1*

The complementary cumulative distribution function (CCDF) of the exceedance r.v. can be written as

$$\begin{aligned}
&\mathbb{P}\left\{|\bar{G}_m| \geq g\right\} \\
&= \mathbb{P}\Big\{|G_m| \geq g \,\Big|\, |G_m| > \eta_{m-1}\Big\}, \forall g \geq \eta_{m-1} && (39)\\
&= \mathbb{P}\Big\{|G_m| \geq \eta_{m-1} + y \,\Big|\, |G_m| > \eta_{m-1}\Big\}, \forall y \triangleq g - \eta_{m-1} \geq 0 && (40)\\
&= \frac{1 - F_{|G_m|}(\eta_{m-1} + y)}{1 - F_{|G_m|}(\eta_{m-1})} = e^{-\frac{g - \eta_{m-1}}{\beta_m}}. && (41)
\end{aligned}$$

From (41), the PoT gradients is distributed as exponential r.v. with location η_{m-1}. Hence, the r.v. $|\bar{G}_m| - \eta_{m-1}$ is exponentially distributed. Consequently, the threshold can be calculated from (6) after proper shifting.

C Proof of Lemma 3 for the Convergence Analysis

Let $\bar{f} : \mathbb{R}^d \to \mathbb{R}$ be a function that is required to be minimized. This function can be a convex or non-convex L_0-smooth function (Karimireddy et al., 2019). Also, the expected value of the stochastic gradient vector equals the oracle gradient, and the second moment of the stochastic gradient is upper bounded by some constant, i.e., σ_0^2.

Let us start first with the assumption that the genie-aided distribution for the amplitude of the stochastic gradient, $F_G(g)$, is known.[9] Hence, the threshold η is calculated as in Equation (5) for some compression ratio δ.[10] After applying the threshold based compression operator $\mathbb{C}_\eta$, the number of non-zero gradients in the sparsified vector is a r.v. distributed as binomial distribution with number of trials d and success probability δ. Hence, the expected number of non-zero gradients matches that of Top$_k$, i.e., $\mathbb{E}\left\{\|\mathbb{C}_\eta\{\mathbf{g}\}\|_0\right\} = \delta d = k$. Therefore, the threshold based compression technique, designed to keep the k largest gradients in magnitude, has the

[9]The genie-aided distribution assumption is relaxed later.

[10]Note, the genie-aided distribution of gradients' amplitude, $F_G(g)$, is not similar to the oracle gradient, $\nabla\bar{f}(\mathbf{x}_{\{i\}})$.

same k-contraction property of Top$_k$ on average

$$\begin{aligned}\mathbb{E}\left\{\|\mathbb{C}_\eta\{\mathbf{g}\}-\mathbf{g}\|_2^2\right\} &= \mathbb{E}\left\{\|\mathbb{T}_k\{\mathbf{g}\}-\mathbf{g}\|_2^2\right\} \\ &\leq (1-\delta)\,\mathbb{E}\left\{\|\mathbf{g}\|_2^2\right\}. \end{aligned} \tag{42}$$

From (42) and Theorem II in (Karimireddy et al., 2019) for compressed SGD adopted with the EC technique, we have

$$\begin{aligned}\min_{i\in[I]}\mathbb{E}\{\left\|\nabla\bar{f}\left(\mathbf{x}_{\{i\}}\right)\right\|_2^2\} \leq \frac{4(\bar{f}(\mathbf{x}_0)-\bar{f}^*)+L_0\,\sigma_0^2}{2\sqrt{I+1}} \\ +\frac{4\,L_0^2\,\sigma_0^2\,(1-\delta)}{\delta^2\,(I+1)}, \end{aligned} \tag{43}$$

where $\bar{f}^*$ is a minimum value for the function $\bar{f}$, and I is the number of iterations over which the function is minimized. Therefore, the rate of convergence of the threshold based scheme with genie-aided distribution coincides with that of Top$_k$ designed with the same compression ratio δ in remark 4 in (Karimireddy et al., 2019). In other words, after $I > \mathbb{O}\left(1/\delta^2\right)$ iteration, the thresholding scheme coincides with the SGD convergence rate.

Now let us move to a more realistic case where we do not know the genie-aided distribution of the gradients. Indeed, there can be a discrepancy between the original and estimated distribution $\hat{F}_G(g)$, which weakens the assumption of SID. In this case, the threshold is estimated as $\hat{\eta}=\hat{F}_G^{-1}(1-\delta)$, leading to an error in the resulting average compression ratio, $\hat{\delta}\triangleq\hat{k}/d$, quantified as

$$\hat{\delta}-\delta \triangleq \frac{1}{d}\left(\mathbb{E}\left\{\|\mathbb{C}_{\hat{\eta}}\{\mathbf{g}\}\|_0\right\}-\mathbb{E}\left\{\|\mathbb{C}_\eta\{\mathbf{g}\}\|_0\right\}\right) \tag{44}$$

$$= F_G\left(\eta(\delta)\right)-F_G(\hat{\eta}(\delta)). \tag{45}$$

In Algorithm 1, the number of thresholding stages are adapted such that

$$\left|\hat{\delta}-\delta\right| \leq \epsilon\,\delta, \qquad 0\leq\epsilon<1\,. \tag{46}$$

Hence, the actual compression ratio can be bounded as

$$\delta\,(1-\epsilon)\leq\hat{\delta}\leq\delta\,(1+\epsilon)\,. \tag{47}$$

For $\hat{\delta}\geq\delta$, the proposed scheme convergences with a rate faster than that of Top$_k$, as the total number of iterations required to reach the SGD's rate is $I>\mathbb{O}\left(\frac{1}{\delta^2\,(1+\epsilon)^2}\right)$, which is smaller than that required for Top$_k$. The reason is that the proposed scheme, in this case, has a better contraction property on average. On the other hand, for $\hat{\delta}\leq\delta$, after number of iterations $I>\mathbb{O}\left(\frac{1}{\delta^2\,(1-\epsilon)^2}\right)$, the proposed scheme coincides with the SGD convergence rate, requiring more iterations than Top$_k$. In Lemma 3, we report only the worst-case convergence rate, requiring more iterations.

D Experimental Specifications

Cluster 1 - Dedicated Environment

- 8 nodes per experiment
- GPUs per node: 1 × Tesla V100-SXM2 with 16GB of GPU memory
- GPU inter-connection: traversing PCIe and the SMP interconnect between NUMA nodes
- CPU: Intel(R) Xeon(R) Silver 4112 CPU @ 2.60GHz, 16 cores
- System memory: 512 GiB
- Ethernet: 25 Gbps SFI/SFP+ - Ethernet
- Network Topology: Star network topology
- OS: Ubuntu 18.04 + Linux Kernel v4.15
- Environment: Horovod's Docker container on DockerHub
- Software: PyTorch 1.1.0, Horovod 0.16, and OpenMPI v4.0

Cluster 2 - Shared Environment

- 1 node per experiment
- GPUs per node: 8 × Tesla V100-SXM2 with 32GB of GPU memory
- GPU inter-connection: traversing PCIe and the SMP interconnect between NUMA nodes
- CPU: Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz, 16 cores
- System memory: 512 GiB
- Ethernet: 100 Gbps - InfiniBand
- Network Topology: Fat Tree topology
- OS: CentOS 7.7 + Linux Kernel v3.10
- Environment: Miniconda 4.3
- Software: PyTorch 1.3, Horovod 0.18, and OpenMPI 4.0

D.1 Further Experimental and Evaluation Details

Here, we present more details on our experimental settings, benchmarks, hyper-parameters, etc. First, we describe the three benchmarks used in this work which covers three commonly used ML tasks in practice. The benchmarks also cover both RNN and CNN architectures.

Image Classification: We studied ResNet20 anf VGG16 on Cifar10, and ResNet-50 and VGG19 on ImageNet. Cifar10 consists of 50,000 training images and 10,000 validation images in 10 classes (Krizhevsky, 2009), while ImageNet contains over 1 million training images and 50,000 validation images in 1000 classes (Deng et al., 2009). We train CIFAR10 models with vanilia SGD (without Momentum) and ImageNet models with Nesterov-momentum SGD following the training schedule in (Gross & Wilber, 2016). The warm-up period is set to 5 epochs for all schemes.

Language Modeling: The Penn Treebank corpus (PTB) dataset consists of 923,000 training, 73,000 validation and

82,000 test words (Marcus et al., 1999). We adopt the 2-layer LSTM language model architecture with 1500 hidden units per layer (Hochreiter & Schmidhuber, 1997). We use Nesterov-momentum SGD with gradient clipping, while learning rate decays when no improvement has been made in validation loss. The warm-up period is 5 epoch out of the 30 epochs.

Speech Recognition: The AN4 dataset contains 948 training and 130 test utterances (AN4). We use DeepSpeech architecture without n-gram language model (Hannun et al., 2014), which is a multi-layer RNN following a stack of convolution layers. We train a 5-layer LSTM of 800 hidden units per layer with Nesterov momentum SGD and gradient clipping, while learning rate anneals every epoch. The warm-up period is 5 epochs out of 150 epochs.

Further Evaluation Details: For training speed-up, we evaluate the speed-up based on the time-to-accuracy of the method that is when it can achieve (or exceed) a certain training accuracy or test perplexity. The target test accuracy is 75% for ResNet20 and 80% for VGG16 on CIFAR-10. The target test perplexity is 105 for PTB benchmark. The target CER is 55 for AN4. We compare no compression, existing and proposed sparsification methods with ratios of $(k = 0.1, 0.01, 0.001)$ using 8 nodes.

E EXTRA EXPERIMENTS, AND RESULTS

In the following, we present more results including more detailed metrics and experimental scenarios. In the following, we refer to 1-stage double Gamma followed by $M-1$ stage Generalized Pareto and multi-stage Generalized Pareto, and multi-stage double exponential are refereed to *SIDCo*-GP, *SIDCo*-P, and *SIDCo*-E respectively.

E.1 Further Metrics and Experimental Scenarios

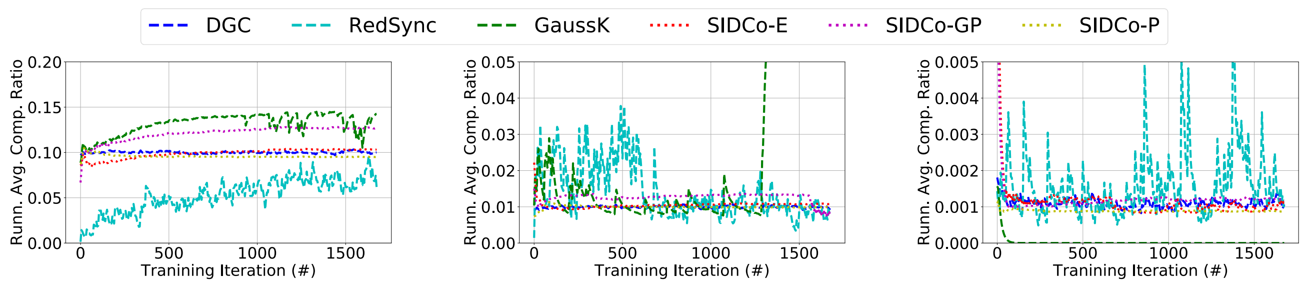
Quality of Estimation Methods: Figure 9 shows the smoothed (or running average) of the compression ratio for all benchmarks and the three ratios (0.1, 0.01, and 0.001) used in the experiments. The results signify the quality of the obtained threshold throughout the training for DGC, RedSync, GaussianKSGD and the three *SIDCo* methods. The results, in general, reinforce our previous observation that *SIDCo* schemes perform quite well and achieve nearly the same threshold quality as of the sampling methods of DGC. *SIDCo* schemes are also significantly better than the other estimation methods (i.e., RedSync and GaussianKSGD). Moreover, other estimation methods (e.g., RedSync and GaussianKSGD) generally results in high oscillations and their over/under-estimation can be up to $\approx \pm 60\times$ the target. We also observe, in few cases, that the multi-stage *SIDCo*-GP (i.e., Gamma-Pareto) results in slight over-estimation which is at most 2 times the target ratio. This could be attributed to the inaccuracies from the first-stage threshold estimation that uses closed-form moment-matching approximation used for fitting the double-Gamma distribution.

To support the observation presented in Figure 18d in which *SIDCo*, unlike all other methods, achieved the target Character Error Rate (CER) because it over-estimated the threshold at early stage of training. In particular Figure 9o shows that *SIDCo*-E algorithm, at the beginning of training, uses the single-stage fitting for the target ratio which leads to threshold over-estimation for few iterations until it settles at the final number of stages. So, thanks to the multi-stage adaptation technique, it can reach to the appropriate number of stages which allows it stay at the target compression ratio. The initial extra-volume at the beginning of training, at this extreme sparsification ratio for this benchmark, leads to significant improvement in accuracy gains and explains the results presented in Figure 3d.

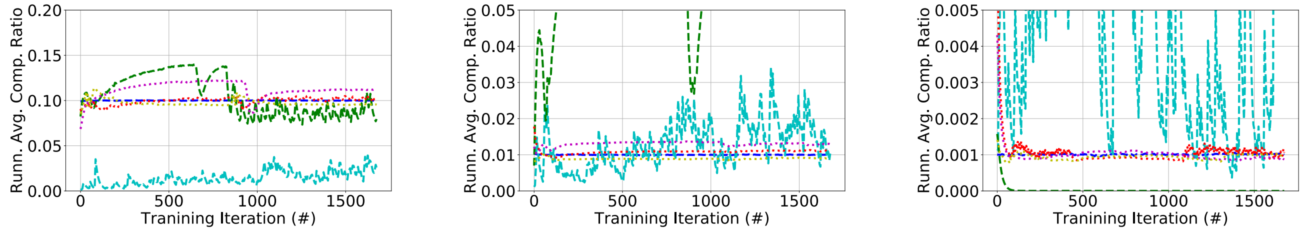
Training Loss: we present the training loss vs run time plots for all benchmarks using all ratios. Figure 10 shows the convergence of all schemes over time and the results in general confirm the speed-up results presented in Section 4.1 and Appendix F. The results highlight the gains in terms of time and accuracy from employing compression over the no-compression. They also signify that most compressors (except for GaussianKSGD and RedSync) achieve same accuracy as Top$_k$ but at lower overhead than Top$_k$.

VGG19 on ImageNet: We also present similar metrics (i.e., smoothed compression ration and training loss vs runtime) for the VGG19 benchmarks in Figure 11. The results in Figure 11a show that all *SIDCo* methods estimate the threshold with high accuracy. They also show that GaussianKSGD miserably fails to estimate the threshold and RedSync experiences significantly high variability. Figure 11a also shows that *SIDCo* methods have noticeably higher speed-ups over all other schemes (esp., Top$_k$, RedSync and GaussianKSGD).

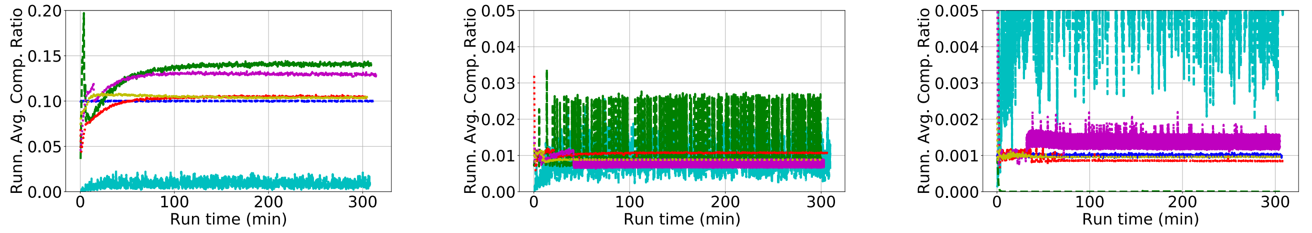
CPU as the Compression Device: In this experiment, instead of using the GPU as the compression target, we use the CPU device as the compression device and report on the average training throughput. Due to the slow speed of the experiment, we only run the experiment for two epochs as we are interested in the throughput numbers. We compare the performance of Top$_k$, DGC and *SIDCo*-E. Figure 12 presents the average training throughput (the first 10 iterations are excluded in the average). First, we note that the throughput on CPU is relatively high for Top$_k$ method which consistently performed the worst when GPU is the target compression device. In contrast, DGC is now performing the worst among all methods due to the slow performance



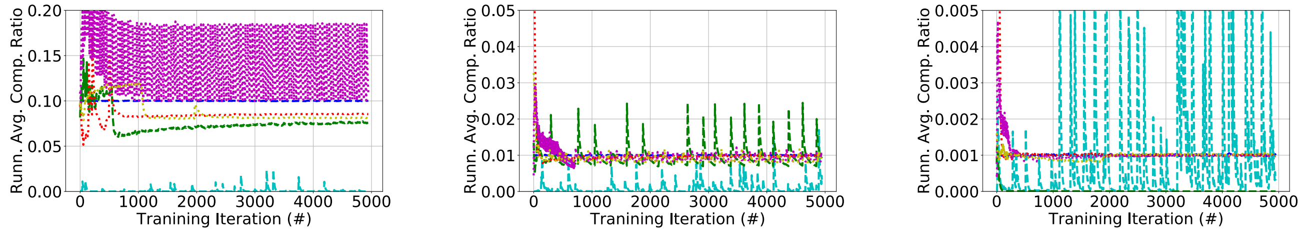
(a) ResNet20 on CIFAR10 - Ratio 0.1. (b) ResNet20 on CIFAR10 - Ratio 0.01. (c) ResNet20 on CIFAR10 - Ratio 0.001.



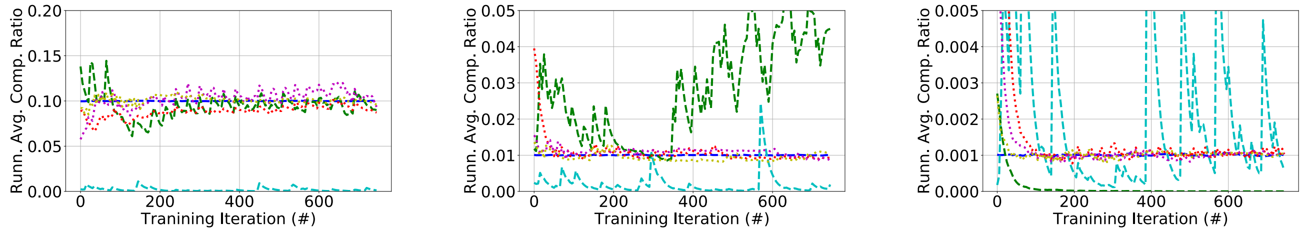
(d) VGG16 on CIFAR10 - Ratio 0.1. (e) VGG16 on CIFAR10 - Ratio 0.01. (f) VGG16 on CIFAR10 - Ratio 0.001.



(g) ResNet50 on ImageNet - Ratio 0.1. (h) ResNet50 on ImageNet - Ratio 0.01. (i) ResNet50 on ImageNet - Ratio 0.001.



(j) PTB on LSTM - Ratio 0.1. (k) LSTM on PTB - Ratio 0.01. (l) LSTM on PTB - Ratio 0.001.



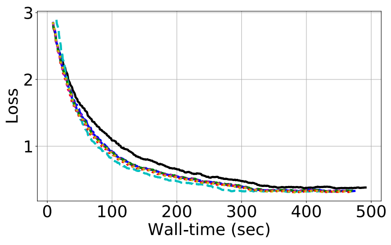
(m) LSTM on AN4 - Ratio 0.1. (n) LSTM on AN4 - Ratio 0.01. (o) LSTM on AN4 - Ratio 0.001.

Figure 9. Smoothed compression ratio for all benchmarks at different ratios.

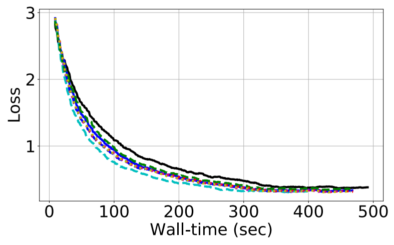
of random sampling on CPU device. On the other hand, *SIDCo* consistently performs the best even on CPU as the target device. These results are not surprising as it closely matches the observations from the micro-benchmark results (Appendix E.2).

Full ImageNet training on Multi-GPU node: In Figure 13, we present the results for training both ResNet50 and VGG19 on ImageNet fully for 90 epochs using a single node equipped with 8 Nvidia-V100 32GB GPUs in the shared cluster presented in Appendix D. Each allocation of

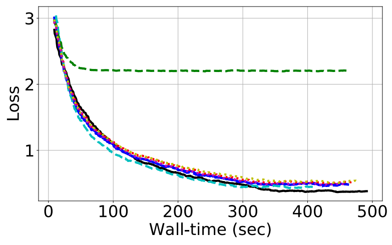
NoComp
Topk
DGC
RedSync
GaussK
SIDCo-E
SIDCo-GP
SIDCo-P



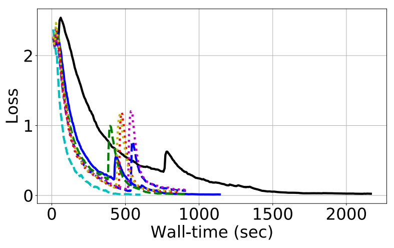
(a) ResNet20 on CIFAR10 - Ratio 0.1.



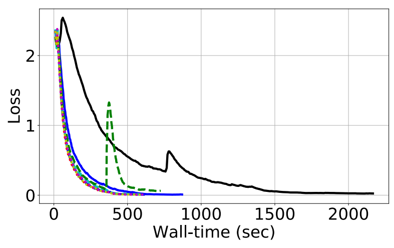
(b) ResNet20 on CIFAR10 - Ratio 0.01.



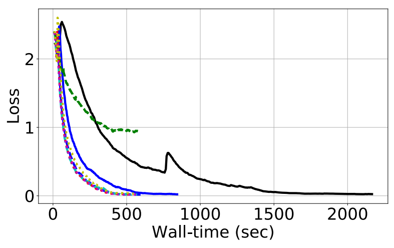
(c) ResNet20 on CIFAR10 - Ratio 0.001.



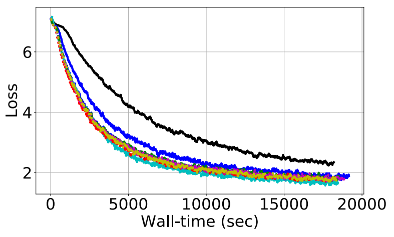
(d) VGG16 on CIFAR10 - Ratio 0.1.



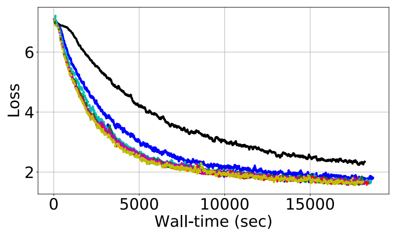
(e) VGG16 on CIFAR10 - Ratio 0.01.



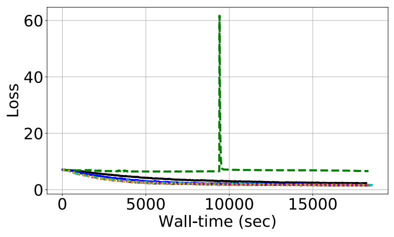
(f) VGG16 on CIFAR10 - Ratio 0.001.



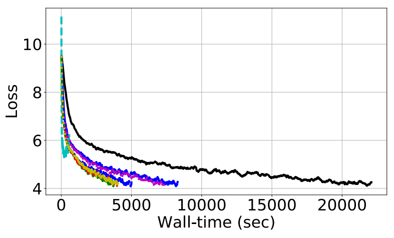
(g) ResNet50 on ImageNet - Ratio 0.1.



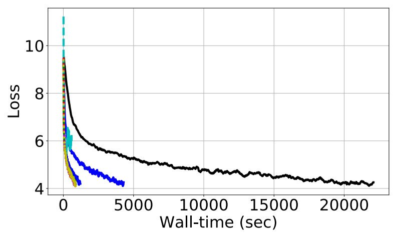
(h) ResNet50 on ImageNet - Ratio 0.01.



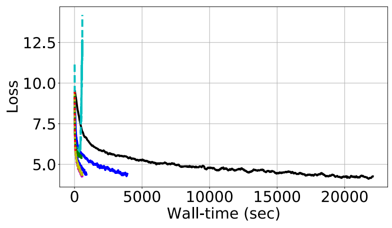
(i) ResNet50 on ImageNet - Ratio 0.001.



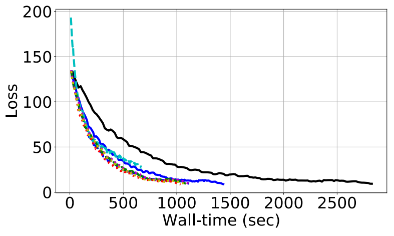
(j) PTB on LSTM - Ratio 0.1.



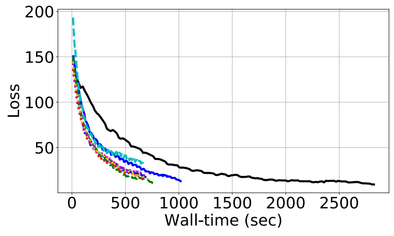
(k) LSTM on PTB - Ratio 0.01.



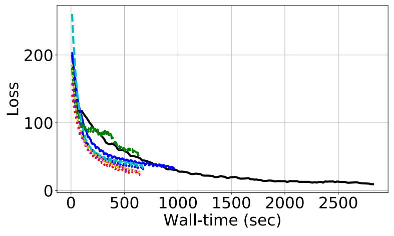
(l) LSTM on PTB - Ratio 0.001.



(m) LSTM on AN4 - Ratio 0.1.



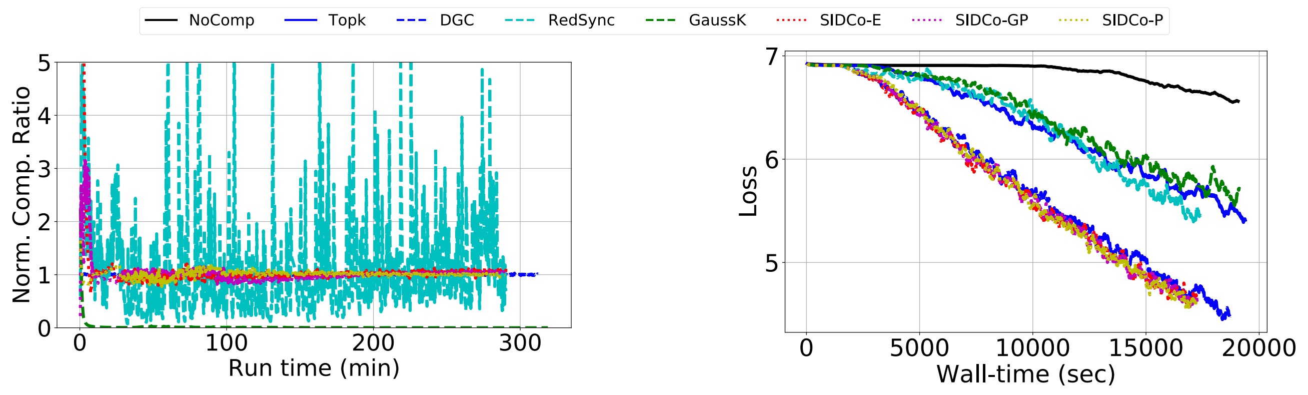
(n) LSTM on AN4 - Ratio 0.01.



(o) LSTM on AN4 - Ratio 0.001.

Figure 10. Smoothed training loss vs wall run-time for all benchmarks at different target sparsity ratios.

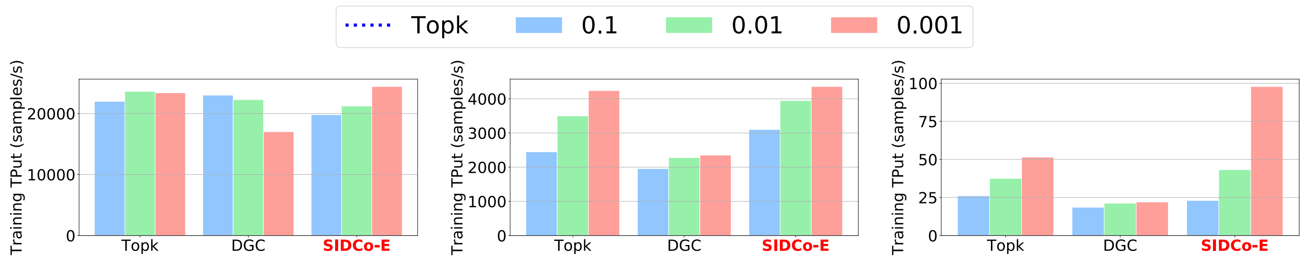
a node in the shared cluster is limited to 24 hours of run-time. We use compression ratio of 0.1 for ResNet50 and 0.01 for VGG19. Figure 13a and 13d show the top-1 test accuracy at the end of the training either due to finishing the 90 epochs or allocation is revoked. They shows that that compression can achieve the same or higher accuracy than no-compression baseline. Also, in case of VGG19, compression speed-ups allow the training to converge faster and hence the higher accuracy. Figure 13b and Figure 13e show the training throughput and that all methods super-



(a) Smoothed compression ratio.

(b) Training loss.

Figure 11. Performance metrics of training VGG19 on ImageNet using ratio of 0.001.



(a) ResNet20 on CIFAR10 (TPut).

(b) VGG16 on CIFAR10 (TPut).

(c) LSTM on PTB (TPut).

Figure 12. Throughput when CPU is the compression device: (a) ResNet20 , (b) VGG16 and (c) LSTM-PTB.

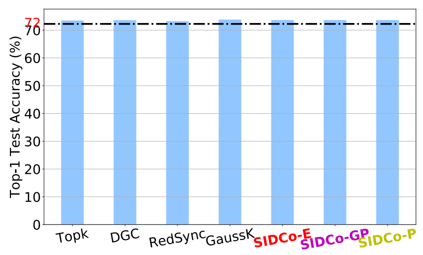
sedes Top$_k$. Moreover, *SIDCo* schemes achieve higher throughput than DGC and Top$_k$. Finally, Figure 13c and Figure 13f show the estimation quality and they show that the quality is very bad for Gaussian-based fitting methods while *SIDCo* schemes can achieve same estimation quality as of the sampling of DGC.

E.2 Compression Complexity of DNN models

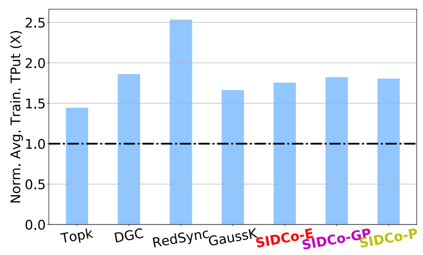
Compression Overhead of Real Models: In Figure 14 and Figure 15, we present the compression speed-up over Top$_k$ and the latency overhead for some models including ResNet20, VGG16, ResNet50 and RNN-LSTM used in training CIFAR10, ImageNet and PTB datasets, respectively. The results confirms the results, presented earlier, for VGG16, where Threshold-based methods including *SIDCo* outperforms Top$_k$ and DGC both on GPU and CPU as target compression device over all models in comparison. The results also show that DGC outperforms Top$_k$ on the GPU device while Top$_k$ outperforms DGC on the CPU device. Overall, for flexibility reasons and the compatibility with various devices, both Top$_k$ and DGC are not preferable.

E.3 Compression Complexity using Synthetic Gradients Vectors of Different Sizes

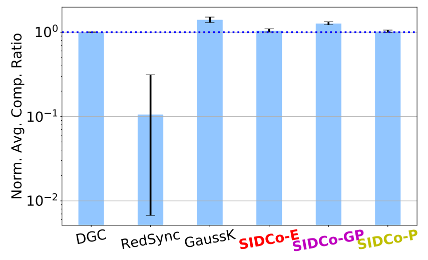
Here, we run the micro-benchmark using synthetic gradient vectors initialized based on input size of (0.26, 2.6, 26, 260) Million elements which is equivalent to $\approx (1, 11, 114, 1140)$ MBytes of gradient data sent in each iteration, respectively. We aim to measure the performance of each compressor in terms of the speed-up over Top$_k$ and latency for wide range of gradient sizes. The results match the former observations on DNN models of different sizes. In particular, Figure 16 shows the speed-up over Top$_k$ on GPU and CPU for each size of the synthetic gradient vectors. We again can observe that on GPU, all methods are faster than Top$_k$ and all threshold estimation methods achieve higher speed-ups over DGC and nearly same speed-ups among each other which is attributed to the slow performance of Top$_k$ (or sorting) operations on GPU. On the CPU, in contrary, we observe that DGC is the slowest method and Top$_k$ excels over it which is attributed to slow performance of random sampling on CPU. Threshold estimation methods maintains same speed-ups on both GPU and CPU (but with relatively different compression times on CPU and GPU).



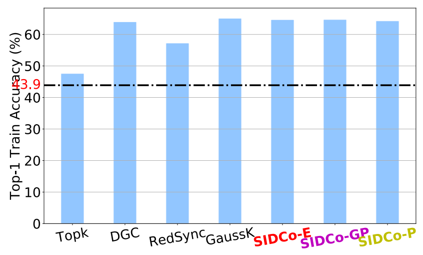
(a) Training accuracy.



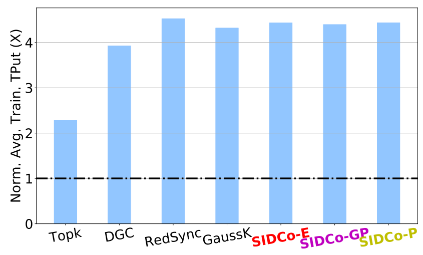
(b) Training Throughput.



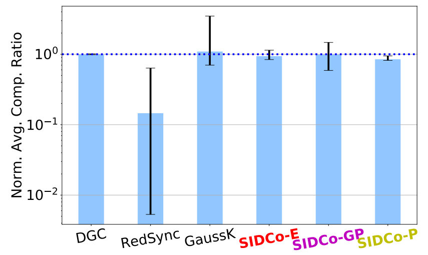
(c) Estimation Quality.



(d) Training accuracy.

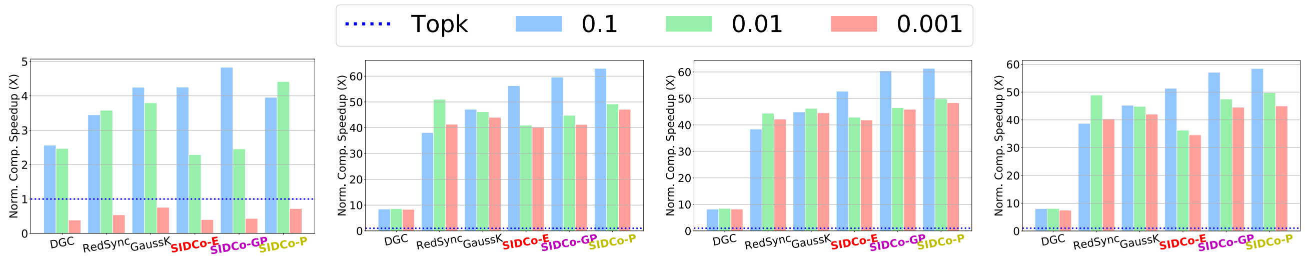


(e) Training Throughput.

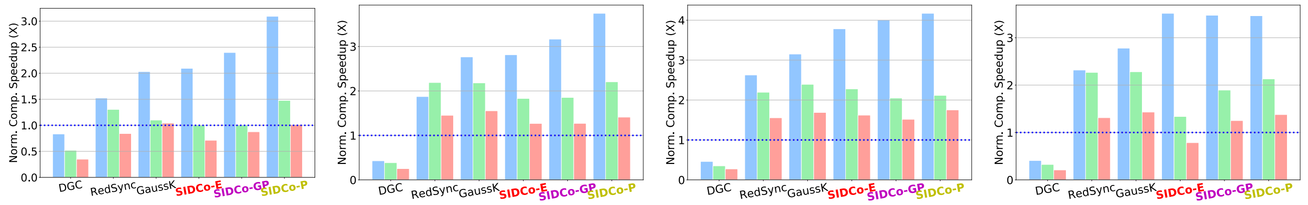


(f) Estimation Quality.

Figure 13. Training Performance of ImageNet on ResNet50 [(a), (b), (c)] and VGG19 [(d), (e), (f)] using the multi-GPU node.



(a) ResNet20 on GPU (b) VGG16 on GPU (c) ResNet50 on GPU (d) LSTM on GPU

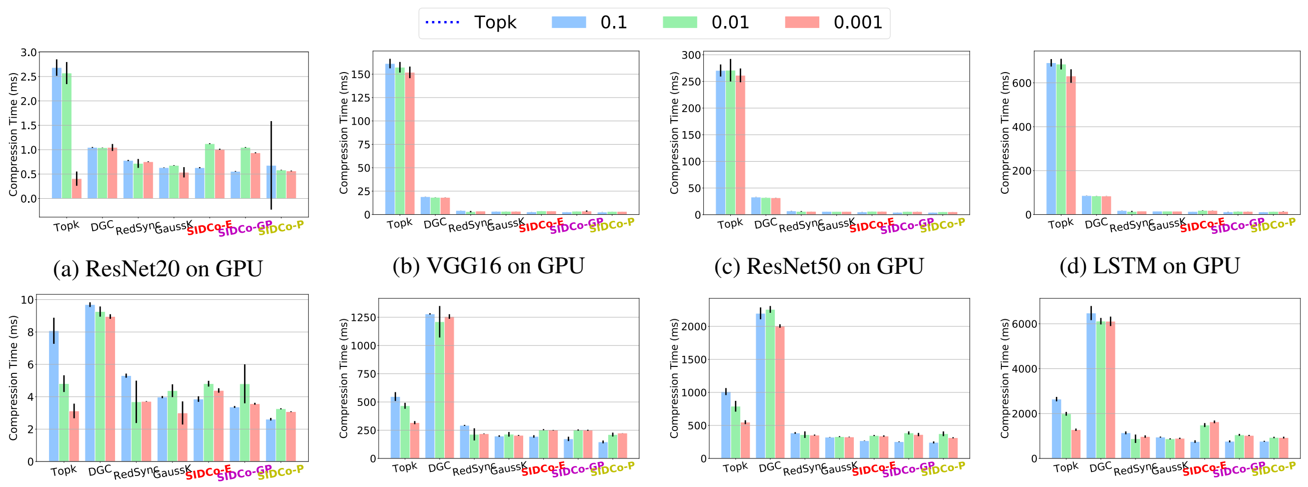


(e) ResNet20 on CPU (f) VGG16 on CPU (g) ResNet50 on CPU (h) LSTM on CPU

Figure 14. Compression speed-up over Top$_k$ of compressing gradient vector of different models using various compressors and ratios on GPU (a,b,c,d) and CPU (e,f,g,h).

F Results of all SIDs

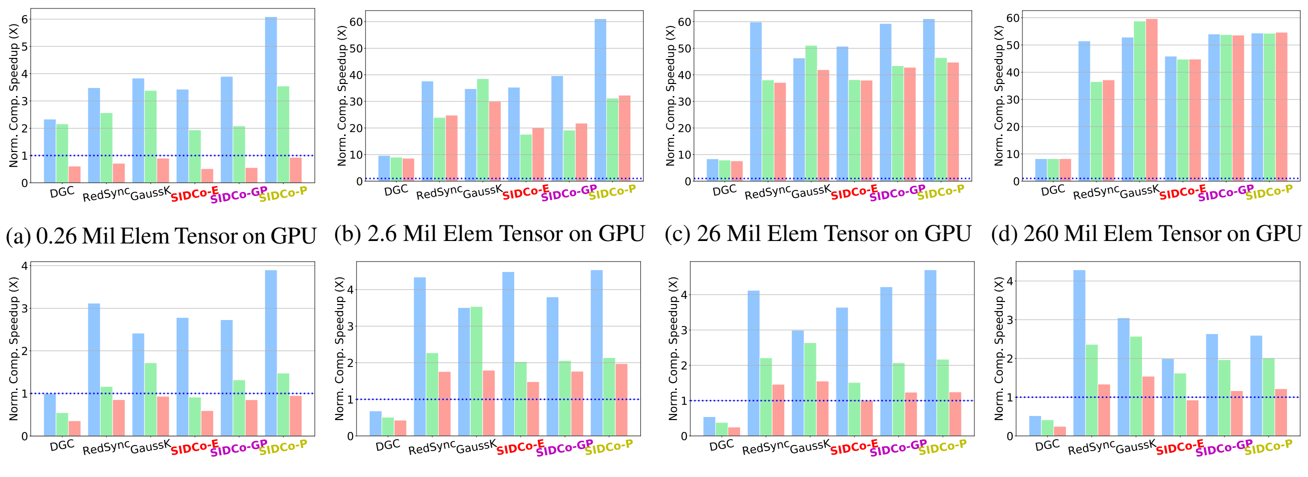
Here, in Figure 18, we include the results for the other two SIDs discussed in Appendix B.3.2, i.e., double Gamma and Generalized Pareto. Note that the two multi-stage SID added here are the 1-stage double Gamma followed by $M-1$ stage of Generalized Pareto and multi-stage Generalized Pareto which are refereed to as *SIDCo*-GP and *SIDCo*-P, respectively. The results and observations are, in general, match the ones we made earlier in Section 4.1 for *SIDCo*-E. However, we observe that, in some cases, *SIDCo*-E achieves slightly better speed-ups compared to *SIDCo*-GP and *SIDCo*-P. This is because of better and slightly lower overhead estimation of the exponential-based threshold which requires only the calculation of the mean of the gradient vector (Algorithm 1). Specifically, in these cases, *SIDCo*-GP which achieves on average the target compression ratio but it tends to have slightly higher variance in terms of the estimation quality (e.g., Figure 18c and Figure 18c). Hence, while variance might be a problem, if it is within the pre-defined tolerance range from the target ratio (ϵ_L,ϵ_H), the impact on the performance would be negligible.



(a) ResNet20 on GPU (b) VGG16 on GPU (c) ResNet50 on GPU (d) LSTM on GPU

(e) ResNet20 on CPU (f) VGG16 on CPU (g) ResNet50 on CPU (h) LSTM on CPU

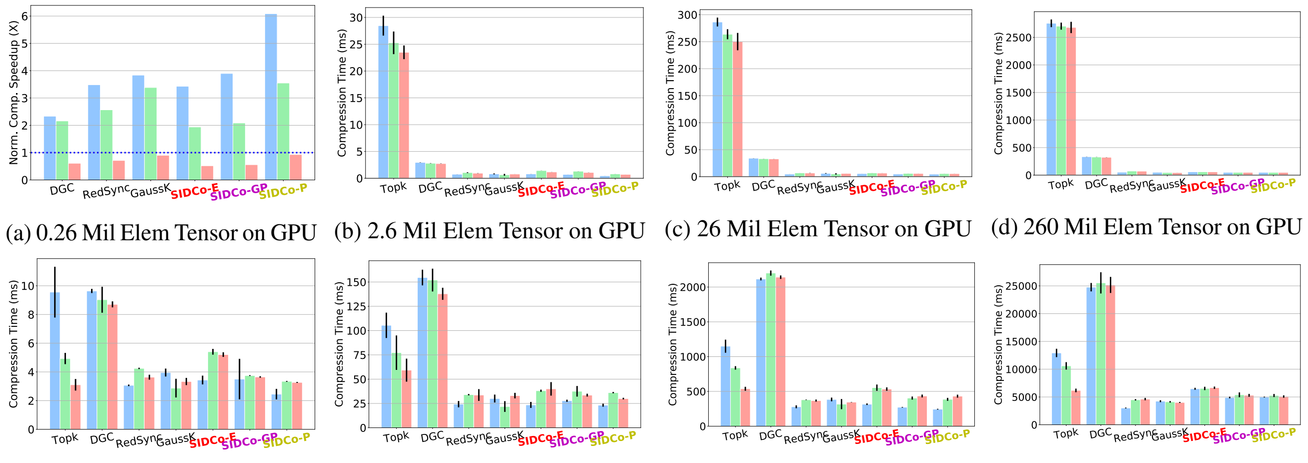
Figure 15. Compression latency of different models using various compressors and ratios on GPU (a,b,c,d) and CPU (e,f,g,h).



(a) 0.26 Mil Elem Tensor on GPU (b) 2.6 Mil Elem Tensor on GPU (c) 26 Mil Elem Tensor on GPU (d) 260 Mil Elem Tensor on GPU

(e) 0.26 Mil Elem Tensor on CPU (f) 2.6 Mil Elem Tensor on CPU (g) 26 Mil Elem Tensor on CPU (h) 260 Mil Elem Tensor on CPU

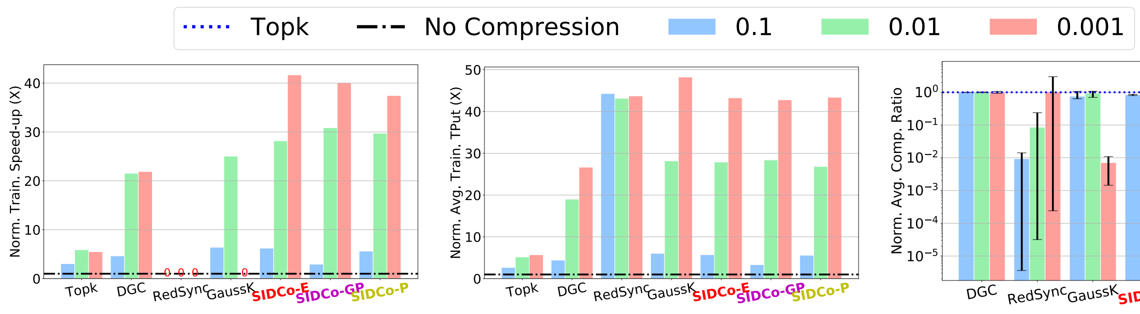
Figure 16. Compression speedups over Top$_k$ of synthetic tensors using various compressors and ratios on GPU (a,b,c,d) and CPU (e,f,g,h).



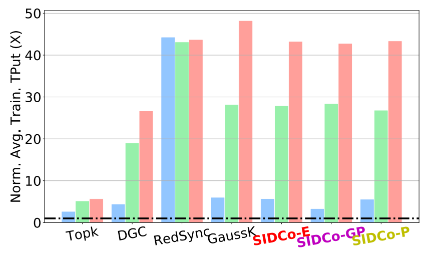
(a) 0.26 Mil Elem Tensor on GPU (b) 2.6 Mil Elem Tensor on GPU (c) 26 Mil Elem Tensor on GPU (d) 260 Mil Elem Tensor on GPU

(e) 0.26 Mil Elem Tensor on CPU (f) 2.6 Mil Elem Tensor on CPU (g) 26 Mil Elem Tensor on CPU (h) 260 Mil Elem Tensor on CPU

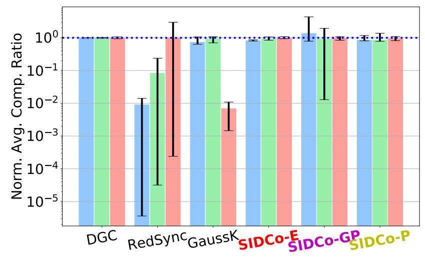
Figure 17. Compression latency of synthetic tensors using various compressors and ratios on GPU (a,b,c,d) and CPU (e,f,g,h).



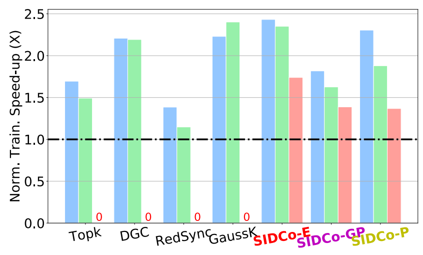
(a) LSTM-PTB (Speed-up).



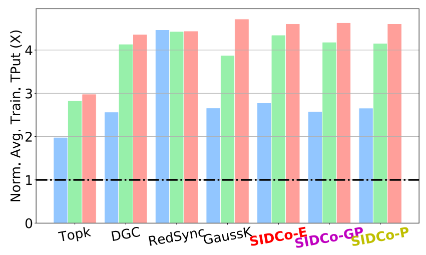
(b) LSTM-PTB (Throughput).



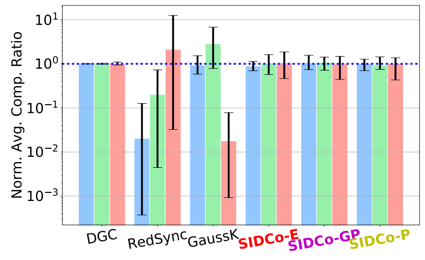
(c) LSTM-PTB (Est. Quality).



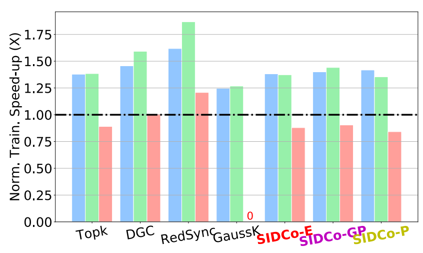
(d) LSTM-AN4 (Speed-up).



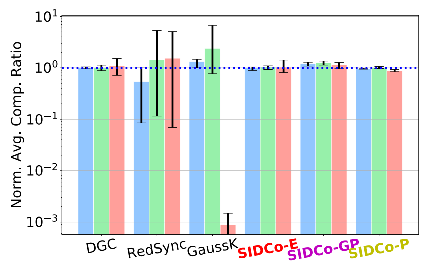
(e) LSTM-AN4 (Throughput).



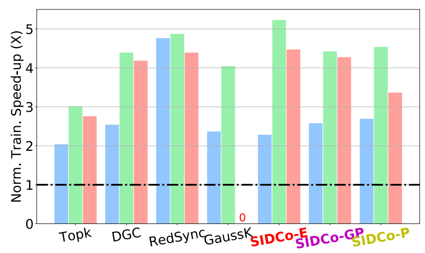
(f) LSTM-AN4 (Est. Quality).



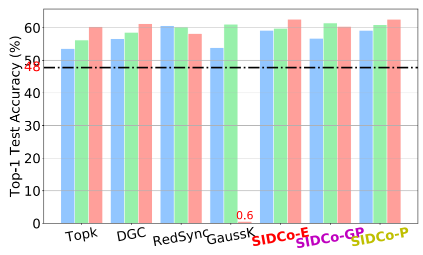
(g) ResNet20-CIFAR10 (Speedup).



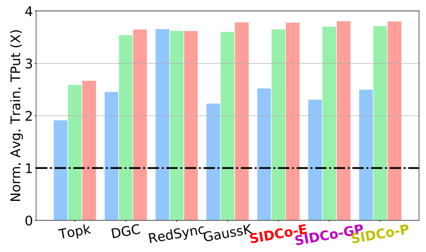
(h) ResNet20-CIFAR10 (Est. Quality).



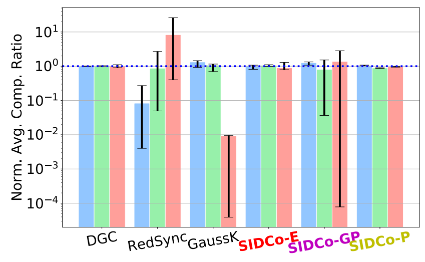
(i) VGG16-CIFAR10 (Speedup).



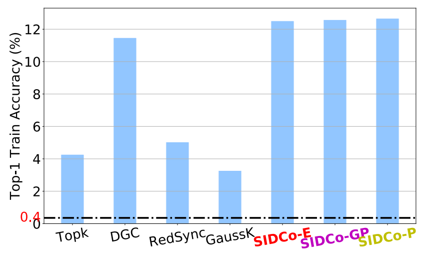
(j) ResNet50-ImageNet (Accuracy).



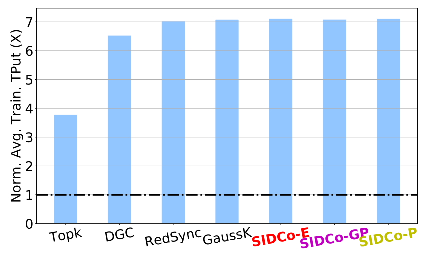
(k) ResNet50-ImageNet (Throughput).



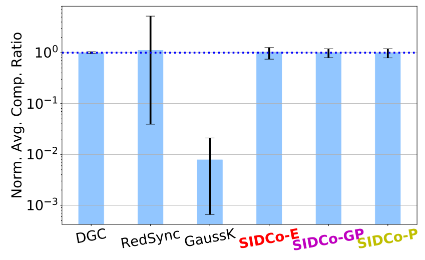
(l) ResNet50-ImageNet (Est. Quality).



(m) VGG19 on ImageNet (Accuracy).



(n) VGG19 on ImageNet (Throughput).



(o) VGG19 on ImageNet (Est. Quality).

Figure 18. Performance of using 8 nodes for training LSTM on PTB [(a),(b),(c)], a LSTM on AN4 [(d),(e),(f)], CIFAR10 on ResNet20 [(g),(h)] and VGG16 [(i)], and training ResNet50 [(j), (k), (l)] and VGG19 [(m), (n), (o)] on ImageNet dataset.