Rethinking gradient sparsification as total error minimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Gradient compression is a widely-established remedy to tackle the communication bottleneck in distributed training of large deep neural networks (DNNs). Under the error-feedback framework, Top-k sparsification, sometimes with k as little as 0.1% of the gradient size, enables training to the same model quality as the uncompressed case for a similar iteration count. We find that, from the optimization perspective, Top-k is the communication-optimal sparsifier given a per-iteration k element budget. We argue that to further the benefits of gradient sparsification, especially for DNNs, a different perspective is necessary — one that moves from per-iteration optimality to consider optimality for the entire training.

We identify that the *total error* – the sum of the compression-errors for all iterations – encapsulates sparsification throughout training. Then, we propose a communication-complexity model that captures minimizing the total error under a communication budget for the entire training. We find that the *hard-threshold sparsifier* – a variant of the Top-k sparsifier with k determined by a constant hard-threshold – is the optimal sparsifier for this model. Motivated by this, we provide convex and non-convex convergence analyses for the hard-threshold sparsifier with error-feedback. Unlike with Top-k sparsifier, we show that hard-threshold has the same asymptotic convergence and linear speedup property as SGD in the convex case, and has no impact of data-heterogeneity in the non-convex case. Our diverse experiments on various DNNs and a logistic regression model demonstrate that the hard-threshold sparsifier is more communication-efficient than Top-k.

1 Introduction

With the emergence of huge DNNs consisting of hundreds of millions to billions of parameters [43, 10], distributed data-parallel training [57] is an increasingly important workload. As the training process typically spans several worker nodes that periodically exchange the local gradient vectors at each iteration of the optimizer (e.g., SGD), communication among workers remains in many cases a main performance bottleneck [39, 28, 36].

Lossy gradient compression techniques are becoming a common approach to rein in communication efficiency [53]. In particular, sparsification, which sends only a subset of gradient coordinates (e.g., Top-k [3, 6] sends the k largest gradient coordinates by magnitude in each iteration), may greatly reduce data volumes and thus speed up training. However, due to its lossy nature, compression raises a complex trade-off between training performance and accuracy. For instance, Agarwal et al. [2] note that training ResNet-18 on CIFAR-100 using sparsification speeds up training significantly ($3.6\times$) but it also degrades final accuracy by 1.5%. On the other hand, Lin et al. [33] reports a $500\times$ data reduction via sparsification under deep gradient compression (DGC) for ResNet-50 on ImageNet while preserving same final accuracy when adopting a carefully-tuned warmup phase.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

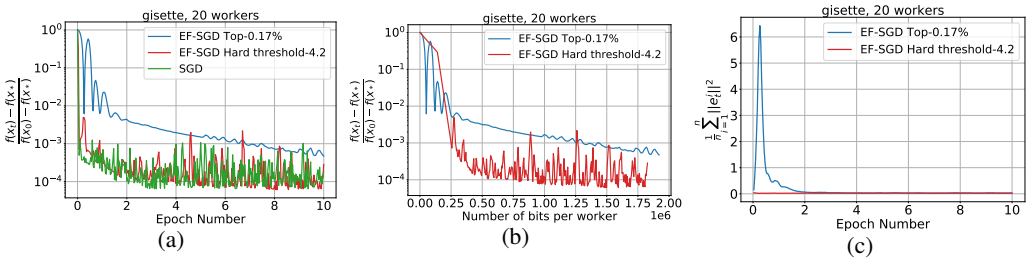


Figure 1: Convergence of Top-k and Hard-threshold for a logistic regression model on `gisette` LIBSVM dataset with 20 workers: (a) Functional suboptimality vs Epochs; (b) Functional suboptimality vs bits communicated; (c) Error-norm vs Epochs. Hard-threshold converges as fast as SGD and much faster than Top-k because of a smaller total-error than Top-k.

The vast literature on gradient compression largely considers a fixed communication budget per iteration, while leaving it up to practitioners to grapple with specifying an additional hyper-parameter that determines the degree of compression before training begins. Meanwhile, recent adaptive Top-k sparsifiers [2, 56] empirically demonstrate that tuning the degree of compression in different phases of DNN training yields a more communication-efficient scheme than a fixed communication scheme (e.g., a static k for Top-k). However, these works lack a theoretical framework proving that adaptive compression enjoys better convergence guarantees than the fixed compression scheme.

This raises a fundamental question: *Given a fixed communication budget, is there a provably better communication scheme than fixed per-iteration compressed communication?* In this paper, we first observe that Top-k is the communication-optimal sparsifier for a fixed per-iteration communication budget (§4.3). Then, our insight is that by adopting a different perspective that accounts for *the effect of sparsification throughout training*, a more efficient communication scheme is possible under a revised notion of optimality that considers an overall communication budget (instead of a per-iteration budget).

Consequently, we consider sparsification using the *error-feedback* (EF) mechanism [45, 6], a delayed gradient component update strategy that is instrumental for the convergence of sparsifiers. Let e_t denote the error arising due to sparsification at iteration t. In error-feedback, this error is added to the gradient update at iteration $t+1$. We identify that the term affecting the non-convex convergence in EF-SGD is the *total-error:* $\sum_t \|e_t\|^2$ [29, 46].

Directly minimizing the total-error is not possible; thus, Top-k minimizes $\|e_t\|^2$ at each iteration. We argue that it is possible to focus on the *sum of* $\|e_t\|^2$ and devise a communication scheme that achieves a smaller total-error than any fixed communication sparsifier. We demonstrate that to achieve this change of perspective, it is sufficient to consider a simple yet practical mechanism that is a natural counterpart of Top-k: the *hard-threshold sparsifier*, which communicates the gradient coordinates with magnitude greater than or equal to a fixed given threshold, $\lambda \geq 0$, in each iteration. Although the two sparsifiers are in an equivalence relation (a given λ corresponds to a k), under the total-error minimization perspective, we adopt a *fixed* threshold, λ, which implies a *variable* k at every iteration.

To illustrate intuitively why this change of perspective yields benefits, consider the following example. Figure 1 shows an experiment where we train on 20 workers a 6,000-parameter logistic regression model on the `gisette` LIBSVM dataset [12] using the Top-k and hard-threshold sparsifiers configured to send the *same data volume*.[1] The loss function is strongly convex and has a unique minimizer, $x^\star$, therefore, a unique optimum, $f(x^\star)$. We see that hard-threshold converges at the same speed as SGD while communicating $\sim 600\times$ less data, whereas Top-k has significantly slower convergence speed. We attribute this to the fact that Top-k has large error accumulation in the initial 500 iterations, while the error magnitude for hard-threshold is less than 0.04 throughout training (cf. Fig. 4c). Our results with DNN training also reflect this insight (§6).

Moreover, the hard-threshold sparsifier has computational benefits over Top-k sparsifier, as hard-threshold's underlying filtering operation requires d comparisons in each iteration, where d is the number of parameters. In contrast, Top-k is a compute-intensive sparsifier (e.g., on CPU, the computational complexity is $\mathcal{O}(d \log_2 k)$ [41]). For GPUs, several optimized implementations are

[1] We train for 10 epochs and set $k = 0.17\%$ for Top-k, and $\lambda = 4.2$ for hard-threshold.

proposed but they rely on the data distribution and are efficient only for a small k [41]. For instance, PyTorch uses Radix select algorithm which has a computational complexity of $\mathcal{O}\left(\lceil b/r \rceil\, d\right)$ where b is the number of bits to represent gradient values and r is the radix size [37].

Finally, while the hard-threshold sparsifier already exists in the literature [47, 17], we are the first to formally study it and theoretically demonstrate its benefits as an adaptive counterpart of Top-k. Moreover, our argument in favor of hard-threshold precisely falsifies the claim by Dryden et al. [16] that stopped its widespread adoption — *a hard-threshold may lead to a degenerate situation when the error-feedback in gradient compression builds up.*

This paper makes the following contributions:

(i) **Communication-complexity model (§4).** We propose a communication complexity model that captures the effects of compression in the entire optimization process. We allow for variable communication in each iteration by only imposing a total communication budget. We show that the hard-threshold sparsifier is the communication-optimal sparsifier in this model.

(ii) **Absolute compressors (§5).** We identify that the hard-threshold sparsifier, along with other existing compressors [14, 39], belongs to the class of *absolute compressors*, which have an absolute bound on the error. Absolute compressors have not been formally studied before with error-feedback. We show that absolute compressors with error-feedback converge for both strongly convex and non-convex loss functions. In both cases, absolute compressors enjoy the same asymptotic convergence with linear speedup (with respect to the number of workers) as no-compression SGD. Notably, δ-contraction operators [45], the class in which Top-k belongs, do not have the same asymptotic convergence and linear speedup property in the strongly convex case. This was previously considered as a drawback of error-feedback in general over unbiased compression [26]. Moreover, for the non-convex case, δ-contraction operators have a worse dependence on δ in the distributed setting with heterogeneous data, while absolute compressors do not have such an anomaly.

(iii) **Experiments (§6).** We conduct diverse experiments on both strongly convex and non-convex (for DNNs) loss functions to substantiate our claims. Our DNN experiments include computer vision, language modeling, and recommendation tasks, and our strongly convex experiment is on logistic regression. We find that the hard-threshold sparsifier is consistently more communication-efficient than the Top-k sparsifier given the same communication budget.

2 Related work

Gradient compression techniques can be broadly classified into quantization [5, 40, 14, 29, 51], sparsification [3, 33, 50], hybrid compressors [7, 47, 16], and low-rank methods [48, 49]. The state-of-the-art compressors are biased δ-contraction operators [48, 33]. We refer to [53] for a recent survey and quantitative evaluation of these techniques.

Error-feedback or memory was first empirically used in [40, 47]. However, [45, 29, 46] were the first to give a convergence analysis of the error-feedback framework, which was extended to the distributed setup in [8, 55]. Recently, [52] proposed *error-reset*, a different form of error-feedback. [26] introduce another alternative by communicating a compressed version of the error.

Communication-optimal compression. [18, 38, 13, 4] devise a communication-optimal compressor by minimizing the worst-case compression-factor[2] under a per-vector communication budget.

Adaptive compression. [50] designed an adaptive sparsifier that minimizes expected sparsity of the compressed vector under a given variance budget. While AdaQS [20] periodically doubles the quantization states in QSGD [5] to reduce the compression-factor, DQSGD [54] sets the number of quantization states proportional to the gradient norm. ACCORDION [2] chooses a low compression ratio if training is in the critical regime, and a high compression ratio otherwise.

Efficient Top-k estimation is a focus of many recent works. While [33] estimates the Top-k threshold on a randomly sampled subset, [42, 1] estimate it by fitting parametric statistical distributions. [22] estimates the threshold every $1,000$ iterations. These works determine a threshold to approximately determine a *fixed* Top-k set, while we use a hard-threshold to determine the k in each iteration.

[2]For a vector x and its compressed vector $\mathcal{C}(x)$, we denote the compression-error as $\mathbb{E}_{\mathcal{C}}[\|x - \mathcal{C}(x)\|^2]$, and compression-factor as $\mathbb{E}_{\mathcal{C}}[\|x - \mathcal{C}(x)\|^2]/\|x\|^2$.

3 Background on Error-Feedback (EF) SGD

Consider the following distributed optimization problem with n workers:

$$\min_{x\in\mathbb{R}^d} f(x) := \frac{1}{n}\sum_{i\in[n]} f_i(x), \tag{1}$$

where $f_i(x) = \mathbb{E}_{z_i\sim\mathcal{D}_i} l(x; z_i)$ such that $l(x; z_i)$ denotes the loss function evaluated on input z_i sampled from $\mathcal{D}_i$, the data-distribution at the ith worker. Let $g_{i,t}$ denote the stochastic gradient computed at i^{th} worker at iteration t such that $g_{i,t} = \nabla f_i(x_t) + \xi_{i,t}$ with $\mathbb{E}[\xi_{i,t}|x_t] = \mathbf{0}$.

Algorithm 1 gives the pseudo-code of compressed EF SGD [45, 29, 6] to solve (1). Let $e_{i,t}$ denote the locally accumulated error at i^{th} worker due to compression from previous steps. Adding this error to the current gradient, $\gamma_t g_{i,t}$ provides the corrected update, $p_{i,t}$. This corrected update is further compressed and exchanged with other machines, and the local error is updated for the next step.

Algorithm 1: Distributed Error-Feedback SGD

Input: $\mathcal{C}$-compressor
Initialize $x_0 \in \mathbb{R}^d$;
for *worker* $i \in [n]$ **in parallel do**
  Initialize error $e_{i,0} = 0$;
  for *iteration* $t = 0, \ldots, T-1$ **do**
    $p_{i,t} \leftarrow e_{i,t} + \gamma_t g_{i,t}$ `/* Incorporate error-feedback into update */`;
    $\Delta_{i,t} \leftarrow \mathcal{C}(p_{i,t})$;
    $e_{i,t+1} \leftarrow p_{i,t} - \Delta_{i,t}$ `/* Update error */`;
    $\bar{\Delta}_t = \frac{1}{n}\sum_{i\in[n]} \Delta_{i,t}$ `/* Exchange and average` $\Delta_{i,t}$ `among workers */`;
    $x_{t+1} \leftarrow x_t - \bar{\Delta}_t$;

3.1 Assumptions

We consider the following general assumptions on the loss function.

Assumption 1. ***(Smoothness)*** *The function, $f_i : \mathbb{R}^d \to \mathbb{R}$ at each worker, $i \in [n]$ is L-smooth, i.e., for every $x, y \in \mathbb{R}^d$ we have, $f_i(y) \leq f_i(x) + \langle \nabla f_i(x), y - x\rangle + \frac{L}{2}\|y - x\|^2$.*

Assumption 2. ***(Global minimum)*** *There exists $x^\star$ such that, $f(x^\star) = f^\star \leq f(x)$, for all $x \in \mathbb{R}^d$.*

Assumption 3. ***((M, σ^2) bounded noise)*** *For every stochastic noise $\xi_{i,t}$, there exist $M, \sigma^2 \geq 0$, such that $\mathbb{E}[\|\xi_{i,t}\|^2 \mid x_t] \leq M\|\nabla f_i(x_t)\|^2 + \sigma^2$, for all $x_t \in \mathbb{R}^d$.*

Remark 1. The above assumption implies $\mathbb{E}[\|g_{i,t}\|^2 \mid x_t] \leq (M + 1)\|\nabla f_i(x_t)\|^2 + \sigma^2$. This general noise model does not uniformly bound the second moment of stochastic gradients as in [29, 55, 7].

Assumption 4. ***((C, ζ^2) bounded similarity)*** *The variance of gradients among nodes is bounded, i.e., there exist constants, $C, \zeta \geq 0$ such that, $\frac{1}{n}\sum_{i\in[n]}\|\nabla f_i(x) - \nabla f(x)\|^2 \leq C\|\nabla f(x)\|^2 + \zeta^2$, for all $x \in \mathbb{R}^d$.*

Remark 2. If all the nodes have the same training data, then all f_i are same, resulting $C, \zeta = 0$. This assumption is an extension from [32, 27], which consider $C = 0$.

For the *convergence of strongly convex functions*, we require an additional assumption as follows.

Assumption 5. ***(μ-strong convexity)*** *The functions, $f_i : \mathbb{R}^d \to \mathbb{R}$ are μ-strongly convex, i.e., there exists $\mu \geq 0$, such that $f_i(y) \geq f_i(x) + \langle \nabla f_i(x), y - x\rangle + \frac{\mu}{2}\|x - y\|^2$, for all $x \in \mathbb{R}^d$.*

Convergence of EF-SGD. The following result shows the convergence of EF-SGD [29] in minimizing general smooth functions for a single worker ($n = 1$) case.

Theorem 1. *[29, 46] Let Assumption 1 and 3 hold. Then Algorithm 1 with a constant stepsize, γ where $\gamma \leq \frac{1}{2L(M+1)}$ and $n = 1$ follows*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(x_t)\|^2 \leq \frac{4(f(x_0) - f^\star)}{\gamma T} + 2\gamma L\sigma^2 + 2L^2\sum_{t=0}^{T-1}\frac{\mathbb{E}\|\gamma g_t + e_t - \mathcal{C}(\gamma g_t + e_t)\|^2}{T}.$$

Remark 3. Theorem 1 is a simplified version of the distributed case for $n > 1$ and quoted to emphasize the effect of compression between the error-corrected update, $\gamma g_t + e_t$, and its compressed form, $\mathcal{C}(\gamma g_t + e_t)$. The term that *solely accounts for the effect of sparsification in the entire training process* is the *total-error*: $\sum_{t=0}^{T-1} \mathbb{E}\|\gamma g_t + e_t - \mathcal{C}(\gamma g_t + e_t)\|^2$.

4 A communication-complexity perspective to sparsification

We now propose a communication complexity model and contrast it with existing communication-optimal strategies. We start with a sparse approximation problem that we encounter in our subsequent discussions.

4.1 A sparse approximation problem

Let $p \in \mathbb{R}^m$ be a given vector. We want to approximate p with a sparse vector, q, that has at most $0 < \tau \leq m$ non-zero elements. Formally, we write the *constrained sparse approximation* problem as:

$$q^\star = \arg\min_{q \in \mathbb{R}^m} \|p - q\|^2 \quad \text{subject to } \|q\|_0 \leq \tau, \tag{2}$$

where $\|\cdot\|_0$ denotes the number of non-zero elements in a vector. Problem (2) and its variants are well studied and arise in signal processing [15, 11] and matrix approximation [44, 9].

Lemma 1. *The solution $q^\star$ to* (2) *is obtained by keeping Top-τ magnitude entries from p and setting the rest to zeros.*

4.2 Minimizing the total-error is not possible

Let $\mathfrak{C}$ denote the class of all compressors. We constrain to the class of deterministic sparsifiers, denoted by $\mathbf{S} \subset \mathfrak{C}$, but one can similarly consider other subclasses in $\mathfrak{C}$. For each $x \in \mathbb{R}^d$, a deterministic sparsifier, $\mathcal{C}_p$ with sparsification parameter, p determines a sparse support set, $S_p(x) \subseteq [d]$ and sparsifies as

$$\mathcal{C}_p(x) = \textstyle\sum_{i \in S_p(x)} x[i] e_i,$$

where e_i denotes the ith standard basis in $\mathbb{R}^d$, and $x[i]$ denotes the corresponding element in x. For example, for hard-threshold sparsifier, $\mathcal{C}_\lambda$, we have $S_\lambda(x) = \{i \mid |x[i]| \geq \lambda\}$. Motivated by Theorem 1 and Remark 3, we now propose the following communication-complexity model:

$$\min_{\mathcal{C} \in \mathbf{S}} \sum_{t=0}^{T-1} \mathbb{E}\|\gamma g_t + e_t - \mathcal{C}(\gamma g_t + e_t)\|^2 \quad \text{subject to } \sum_{t=0}^{T-1} \|\mathcal{C}(\gamma g_t + e_t)\|_0 \leq K, \tag{3}$$

where K is the budget on the number of elements communicated in T iterations. However, solving (3) is intractable since the optimization process is dynamic with new gradient and error depending on the previous gradients and errors, and involves randomness due to stochastic gradients.

4.3 Top-k is communication-optimal for a per-iteration k element budget

To simplify (3), one can focus individually at the error at each iteration. Based on this, we show in Lemma 2 that Top-k has the best compression-error among all sparsifiers under a per-iteration k-element communication budget.

Lemma 2. *Given the gradient g_t and error e_t at iteration t, Top-k sparsifier achieves the optimal objective for the optimization problem:*

$$\min_{\mathcal{C} \in \mathbf{S}} \|\mathcal{C}(\gamma g_t + e_t) - (\gamma g_t + e_t)\|^2 \quad \textit{subject to } \|\mathcal{C}(\gamma g_t + e_t)\|_0 \leq k. \tag{4}$$

Similar to Lemma 1, (4) is solved when $\mathcal{C}(\gamma g_t + e_t)$ contains the k highest magnitude elements of $\gamma g_t + e_t$. That is, when $\mathcal{C}$ is the Top-k sparsifier. Additionally, based on the above model, a per-iteration k-element communication budget (resulting in a total budget of kT elements throughout training), implies that Top-k is performed at each iteration. However, to have a more communication-efficient compression, we require a communication-complexity model that (*i*) better captures total-error in Theorem 1; and (*ii*) allows for adaptive communication, i.e., sends variable data in each iteration.

4.4 A communication-complexity model for adaptive sparsification

Although the total-error cannot be minimized (§4.2), Lemma 2 motivates us to consider a *simplified model* that can capture the total-error. Instead of $(\gamma g_t + e_t)_{t=0}^{T-1}$, we consider a fixed sequence $(a_t)_{t=0}^{T-1}$ and examine the following communication-complexity model:

$$\min_{\mathcal{C}\in\mathbf{S}} \sum_{t=0}^{T-1} \|a_t - \mathcal{C}(a_t)\|^2 \quad \text{subject to } \sum_{t=0}^{T-1} \|\mathcal{C}(a_t)\|_0 \leq K, \tag{5}$$

where $K \in \mathbb{N}$ denotes the total communication budget.

Let $\mathcal{A} \in \mathbb{R}^{dT}$ be formed by stacking $(a_t)_{t=0}^{T-1}$ vertically and consider the following sparse approximation problem:

$$\min_{B\in\mathbb{R}^{dT}} \|\mathcal{A} - B\|^2 \quad \text{subject to } \|B\|_0 \leq K, \tag{6}$$

Note that (6) allows for all B that are formed by stacking $(\mathcal{C}(a_t))_{t=0}^{T-1}$ vertically, for some sparsifier $\mathcal{C}$ satisfying $\sum_{t=0}^{T-1} \|\mathcal{C}(a_t)\|_0 \leq K$. Therefore, the optimal objective for (6) is a lower bound to the optimal objective for (5). Let $\mathcal{A}_{(i)}$ denote the element with i^{th} largest magnitude in $\mathcal{A}$, and assume no two elements have the same magnitude, i.e., $\mathcal{A}_{(i+1)} \neq \mathcal{A}_{(i)}$, for all $i \in [dT]$. Then, $B = \mathcal{C}_\lambda(A)$ with $\lambda \in \left(A_{(K+1)}, A_{(K)}\right]$ contains the Top-K magnitude entries from A, and therefore, by Lemma 1 is optimal for (6). Moreover, since hard-threshold is an element-wise sparsifier, $\mathcal{C}_\lambda(A)$ is equivalent to stacking $(\mathcal{C}_\lambda(a_t))_{t=0}^{T-1}$ vertically. Therefore, $\mathcal{C}_\lambda$ with $\lambda = \left(\mathcal{A}_{(K+1)}, \mathcal{A}_{(K)}\right]$ achieves optimal objective in (5). We formalize it in the following lemma.

Lemma 3. *$\mathcal{C}_\lambda$ is optimal for the communication-complexity model* (5)*. That is, for every budget K, there exists a λ such that $\mathcal{C}_\lambda$ minimizes* (5)*.*

4.5 Discussion

To capture the effect of compression, existing works [5, 45, 29] use a bound on the *compression-factor*, $\max_{x\in\mathbb{R}^d} \mathbb{E}_{\mathcal{C}}\|\mathcal{C}(x)-x\|^2/\|x\|^2$. We formally define them as relative compressors.

Relative Compressor [5, 45]. An operator, $\mathcal{C} : \mathbb{R}^d \to \mathbb{R}^d$ is a *relative compressor* if for all vector, $x \in \mathbb{R}^d$ it satisfies

$$\mathbb{E}_{\mathcal{C}}\|x - \mathcal{C}(x)\|^2 \leq \Omega\|x\|^2, \tag{7}$$

where $\Omega > 0$ is the compression factor and the expectation, $\mathbb{E}_{\mathcal{C}}$, is taken with respect to the randomness of $\mathcal{C}$. *δ-contraction operators* [45, 29] with $\Omega = 1 - \delta$ and $\delta \in (0, 1]$, are special cases of relative compressors.

Top-k is a δ-contraction operator with $\delta = \frac{k}{d}$ [45]. Therefore, by (7), Top-k allows for larger compression-error with larger inputs. Our communication-complexity model demonstrates that this might not necessarily be a good idea. Moreover, with error-feedback, a large error at any iteration has a cascading effect — a large e_t results in a large $\gamma g_t + e_t$, out of which only k/d fraction of the total components are kept by the Top-k strategy. This results in a large e_{t+1} (see §B.2). Figure 1 shows that this *error-buildup* has severe implications on the total-error. On the other hand, hard-threshold performs a variable Top-k in each iteration, and sends an element as soon as its magnitude is bigger than the threshold. This prohibits the error build up.

Comparison with existing communication-optimal compression strategies. Since Ω solely defines the effect of compression in convergence [26, 46], many recent works [18, 38, 13, 4] propose communication-optimal compression strategies by optimizing for Ω under a *communication budget* for each vector, i.e., they propose to solve

$$\min_{\mathcal{C}\in\mathfrak{C}} \max_{x\in\mathbb{R}^d} \frac{\mathbb{E}_{\mathcal{C}}\|\mathcal{C}(x) - x\|^2}{\|x\|^2} \quad \text{subject to } \texttt{Bits}(\mathcal{C}(x)) \leq B, \tag{8}$$

where $\texttt{Bits}(\mathcal{C}(x))$ denotes the number of bits needed to encode $\mathcal{C}(x)$. We stress that while the compression-affected term in Theorem 1 has the sum of compression-errors, the above communication-complexity model only captures the compression-factor.

5 Absolute compressors and their convergence

Motivated by the previous section, we formally define *absolute compressors*—compressors that have an absolute bound on the error.

Absolute Compressor. An operator, $\mathcal{C} : \mathbb{R}^d \to \mathbb{R}^d$ is an *absolute compressor* if there exists a $\upsilon > 0$ such that for all vector, $x \in \mathbb{R}^d$ it satisfies

$$\mathbb{E}_{\mathcal{C}}\|x - \mathcal{C}(x)\|^2 \leq \upsilon^2. \tag{9}$$

In contrast to the relative compressors in (7), the compression-error (or, variance) of absolute compressors is bounded by a constant, independent of x. Based on the above definition, hard-threshold is an absolute sparsifier with $\upsilon^2 = d\lambda^2$. The stochastic rounding schemes with bounded rounding error in [21] are absolute compressors (used for model quantization). Precisely, any rounding scheme, with rounding error bounded by ϵ, is an absolute compressor with $\upsilon^2 = d\epsilon^2$. Similarly, the scaled integer rounding scheme in [39] is an absolute compressor. While this class of compressors existed in the literature, we are the first to provide their convergence result with an EF.

5.1 Convergence results

Inspired by [46], we establish convergence analysis of EF-SGD (Algorithm 1) with absolute compressors. Convergence analysis for the momentum case [55] can be extended in a similar way, but we do not include it for brevity and the existing analyses do not show any benefit over vanilla SGD. Similarly, analysis for error-reset [52] and local updates [7, 52] can also be extended. We provide convergence results for the convex and non-convex case and compare them to that of δ-contraction operators.

We start with a bound on the error. In Algorithm 1, we compress the error-compensated updates, i.e. gradients multiplied by the stepsize. Notably, absolute compressors are not *scale-invariant*, that is, $\mathcal{C}(\alpha x) \neq \alpha\mathcal{C}(x)$, for $\alpha \in \mathbb{R}^+$. This leads them to send more data when $\alpha > 1$, and vice versa. Therefore, we scale the error-threshold, υ by the stepsize at each iteration as $\upsilon_t = \gamma_t\kappa$, for some $\kappa > 0$. The above can be ensured for hard-threshold by setting the threshold as $\gamma_t\lambda$, where λ is a threshold for error-compensated gradients. With this, we arrive at an error bound in Remark 4. We note the scaling is not required if we instead compressed the error-compensated gradients as in [55].

Remark 4. **(Error bound)** Let $\upsilon_t = \gamma_t\kappa$ for some $\kappa > 0$. Then for all $i \in [n], t \in \{0, \dots T-1\}$, we have

$$\mathbb{E}_{\mathcal{C}}[\|e_{i,t+1}\|^2 \mid p_{i,t}] = \mathbb{E}_{\mathcal{C}}\|p_{i,t} - \mathcal{C}(p_{i,t})\|^2 \leq \gamma_t^2\kappa^2.$$

A similar absolute bound for δ-contraction operators requires the bounded gradient assumption [29, 7], but absolute compressors achieve this by design.

5.1.1 Convex convergence

Throughout this section, we denote $P_T := \mathbb{E}[f(\bar{x}_T)] - f^\star$ and $R_T := \|x_T - x^\star\|^2$ for $T \geq 0$, where $\bar{x}_T = \frac{1}{W_T}\sum_{t=0}^{T} w_t x_t$, and $W_T = \sum_{t=0}^{T} w_t$ with $w_t \geq 0$. Further denote $D := \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x^\star)\|^2$. With these notations, we quote the strongly convex ($\mu > 0$) and convex ($\mu = 0$) convergence results for absolute compressors, and compare them with the δ-contraction operators from [8] for distributed case ($n \geq 1$). For exact choices of step-sizes and weights, see §A.4.

Theorem 2. *Let $\mu > 0$ and Assumptions 1, 3, and 5 hold. Then the iterates, $\{x_t\}_{t\geq 0}$ of Algorithm 1 with an absolute compressor, $\mathcal{C}_\upsilon$, a constant stepsize, $\gamma(T)$ with $\gamma(T) \leq \frac{1}{4L(1+2M/n)}$ follow* [3]

$$P_T = \tilde{\mathcal{O}}\left(L(1+M/n)R_0 \exp\left[-\frac{\mu T}{8L(1+2M/n)}\right] + \frac{\sigma^2 + MD}{\mu n T} + \frac{L\kappa^2}{\mu^2 T^2}\right).$$

Remark 5. Under the same setting as in Theorem 2, iterates of Algorithm 1 with δ-contraction operators [8] follow:

$$P_T = \tilde{\mathcal{O}}\left(L(1/\delta + M)R_0 \exp\left[-\frac{\mu T}{28L(2/\delta + M)}\right] + \frac{\sigma^2 + D(1/\delta + M/n)}{\mu T}\right).$$

Remark 5 indicates some key convergence behaviors of δ-contraction operators for strongly convex case:(*i*) The compression slows down the slower decaying $\mathcal{O}(1/T)$ term, as $1/\delta$ occurs in it. (*ii*) They have slower exponential convergence with compression affecting the exponential term. (*iii*) They do not exhibit the linear speedup in the slower decaying $\mathcal{O}(1/T)$ term. The absence of a linear speedup was considered as a disadvantage of error-feedback over unbiased relative compression [26]. As our result for absolute compressors with error-feedback in Theorem 2 has the linear speedup property, we conclude the absence of a linear speedup is a drawback of δ-contraction operators and not error-feedback in general. See further comparison against unbiased compressors [26] in §A.5.

[3] The $\tilde{\mathcal{O}}$ notation hides constants and factors polylogarithmic in the problem parameters.

Theorem 3. *Let $\mu = 0$ and Assumptions 1, 3, and 5 hold. Then the iterates, $\{x_t\}_{t\geq 0}$ of Algorithm 1 with an absolute compressor, $\mathcal{C}_\upsilon$, a constant stepsize, $\gamma(T)$ with $\gamma(T) \leq \frac{1}{4L(1+2M/n)}$ follow*

$$P_T = \mathcal{O}\left(\frac{\sqrt{(\sigma^2+MD)R_0}}{\sqrt{nT}} + \frac{\left(\frac{nL\kappa^2}{\sigma^2+MD}+L(1+M/n)\right)R_0}{T}\right).$$

Remark 6. Theorem 3 holds when both σ^2 and D are not simultaneously zero. Typically, we encounter heterogeneous data settings where $D \neq 0$, and Theorem 3 holds. In case both σ^2 and D are zero, we get $\mathcal{O}(\frac{(L\kappa R_0)^{2/3}}{T^{2/3}})$ convergence.

Remark 7. Under the same setting as in Theorem 3, iterates of Algorithm 1 with δ-contraction operators follow [8]

$$P_T = \mathcal{O}\left(\frac{\sqrt{(\sigma^2+D(1/\delta+M/n))R_0}}{\sqrt{T}} + \frac{L(1/\delta+M)R_0}{T}\right).$$

Remark 7 indicates some key convergence behaviors of δ-contraction operators for convex case: (*i*) The compression slows down the slower decaying $\mathcal{O}(1/\sqrt{T})$ term as $1/\delta$ occurs in it; (*ii*) δ-contraction operators do not have linear speedup in the slower decaying $\mathcal{O}(1/\sqrt{T})$ term.

Designing variance-reduced algorithms by using absolute compressors with EF is orthogonal to this work.

5.1.2 Non-convex convergence

Theorem 4. ***(Non-convex convergence of absolute compressors)*** *Let Assumptions 1 , 3, and 4 hold. Then the iterates, $\{x_t\}_{t\geq 0}$ of Algorithm 1 with an absolute compressor and a constant stepsize $\gamma \leq \frac{n}{2L(M(C+1)+n)}$ follow*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(x_t)\|^2 \leq \frac{4(f(x_0)-f^\star)}{\gamma T} + \frac{2\gamma L(M\zeta^2+\sigma^2)}{n} + 2\gamma^2L^2\kappa^2.$$

Alongside, we compare with the non-convex convergence for δ-contraction operators in distributed setting. The existing analyses tackle this by using a stronger *uniform bounded gradient* assumption [55, 29, 17]. To the best of our knowledge, we are the first to use weaker Assumption 3 and 4, to establish a non-convex convergence analysis of δ-contraction operators in distributed setting. [46] introduced a relaxed Assumption 3 to establish non-convex convergence of δ-contraction operators but only for the single-node case, while [8] extended their result only for μ-strongly convex functions in distributed case.

Theorem 5. ***(Non-convex convergence of δ-contraction operators)*** *Let Assumptions 1, 3, and 4 hold. Then the iterates, $\{x_t\}_{t\geq 0}$ of Algorithm 1 with a δ-compressor and a constant step-size $\gamma \leq \min\{\frac{n}{2L(M(C+1)+n)}, \frac{1}{2L(2/\delta+M)\sqrt{C+1}}\}$ follow*

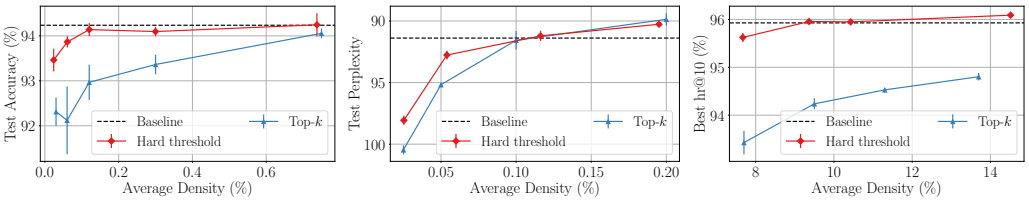
$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(x_t)\|^2 \leq \frac{8(f(x_0)-f^\star)}{\gamma T} + \frac{4\gamma L(M\zeta^2+\sigma^2)}{n} + \frac{8\gamma^2L^2}{\delta}\left(\left(\frac{2}{\delta}+M\right)\zeta^2+\sigma^2\right).$$

The above result implies, in distributed settings with heterogeneous data ($\zeta \neq 0$), δ-contraction operators have an $1/\delta^2$ dependence on δ, as compared to an $1/\delta$ dependence in the homogeneous case ($\zeta = 0$). In contrast, absolute compressors have the same κ^2 dependence on κ in homogeneous and heterogeneous cases. Therefore, we conjecture it is beneficial to use absolute compressors in settings such as federated learning [34], where data heterogeneity is widely encountered.

Remark 8. With appropriate choices of step-size, both absolute compressors and δ-contraction operators with EF-SGD achieve the same $\mathcal{O}(1/\sqrt{nT})$ asymptotic rate of SGD. See Corollary 1 in §A.3 for the full result.

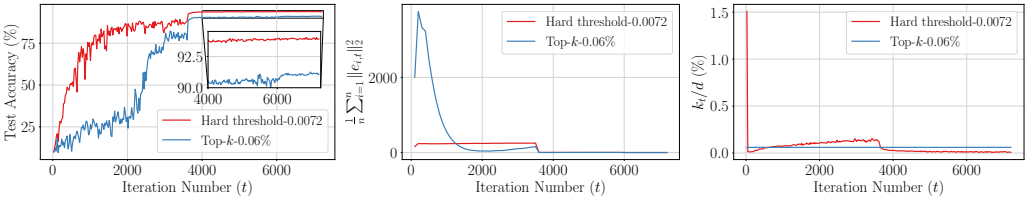
6 Experiments

Experimental setup. We compare Top-k and hard-threshold sparsifiers on image classification, language modelling, and recommendation tasks. We use different optimizers: vanilla SGD, SGD with Nesterov momentum, and ADAM. All experiments were run on an 8-GPU cluster, using `Allgather` the communication primitive. We perform compression in the standard layer-wise fashion [40, 33, 17] and follow the error-feedback strategy used in [48]. For hyper-parameter configuration, comparison



(a) ResNet-18 on CIFAR-10 (b) LSTM on Wikitext (c) NCF on Movielens-20M

Figure 2: **Test metric vs. Data Volume.** For 3 benchmarks, average test quality with std. dev. over 3 runs. The dashed black line denotes the no compression baseline.



(a) (b) (c)

Figure 3: **Convergence of Top-k and Hard-threshold for ResNet-18 on CIFAR10 at 0.06% average density:** (a) Test-accuracy vs Iterations, (b) Error-norm vs Iterations, (c) Density (k_t/d) vs Iterations. $k = 0.06\%$ of d, and $\lambda = 0.0072$. Hard-threshold has better convergence than Top-k because of a smaller total-error.

with entire-model compression, discussion on different error-feedback approaches, experiments without error-feedback, and experiments with logistic regression, we refer to Appendix B.

Test metric vs. Data volume. We tune the sparsification parameters for both sparsifiers such that they send similar total data volumes during training. We use *average density*: $\frac{1}{T}\sum_{t=0}^{T-1}\frac{k_t}{d}$ as a measure of total data volume, where k_t denotes the number of elements transmitted in iteration t. Figure 2 shows the average test quality across three repetitions with different initial random seeds. We observe that fixing the average density, *hard-threshold consistently has better test performance* than Top-k. For ResNet-18 on CIFAR10, we observe that hard-threshold at average density of 0.12% almost achieves the baseline accuracy, and is better than Top-k at 0.75% density ($\sim 6\times$ more total data volume). For LSTM on Wikitext, at an average density of 0.025%, hard-threshold has > 2 better perplexity than Top-k. For NCF on Movielens-20M, hard-threshold has $> 1\%$ better Hit-Rate@10 at all considered average densities.

We now demonstrate that hard-threshold has faster convergence because of a smaller total-error in comparison to Top-k. In Fig. 3, we introspect a run with average density of 0.06% from Fig. 2a. In Fig. 3a, while hard-threshold converges to an accuracy of 93.9%, Top-k achieves 91.1% accuracy. At the same time, in Fig. 3b, we observe large error-accumulation in the initial $1,200$ iterations for Top-k. Consequently, hard-threshold has a significantly lower total-error than Top-k, and therefore has better convergence. This observation about large error accumulation for Top-k is consistent across all our benchmarks (see §B.2).

Comparison against ACCORDION [2]. We compare against the state-of-the-art adaptive sparsifier: ACCORDION [2]. ACCORDION shifts between two user-defined k values: $k_{\max}$ and $k_{\min}$, by using Top-$k_{\max}$ when the training is in a *critical regime*, else using Top-$k_{\min}$. We compare against ACCORDION with threshold $\lambda = \frac{1}{2\sqrt{k_{\min}}}$. With this hard-threshold, we observe improved and more communication-efficient performance over ACCORDION (e.g., for CIFAR10 dataset, up to 0.8% better accuracy with $4\times$ fewer communicated bits, see §B.4).

How to tune the hard-threshold? Based on our results in §5.1.2, we suggest setting the threshold as $\lambda \sim \frac{1}{2\sqrt{k}}$ to achieve better convergence under similar total-data communication as Top-k with parameter k (see further discussion in §C). But it remains an open question how to tune the hard-threshold such that it achieves no-compression baseline performance with the least total-data transmission. We note that, as of now, this question remains unanswered for Top-k as well.

References

[1] A. M. Abdelmoniem, A. Elzanaty, M-S. Alouini, and M. Canini. An Efficient Statistical-based Gradient Compression Technique for Distributed Training Systems. In *MLSys*, 2021.

[2] S. Agarwal, H. Wang, K. Lee, S. Venkataraman, and D. Papailiopoulos. Accordion: Adaptive Gradient Communication via Critical Learning Regime Identification. In *MLSys*, 2021.

[3] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[4] A. Albasyoni, M. Safaryan, L. Condat, and P. Richtárik. Optimal gradient compression for distributed and federated learning. *arXiv preprint arXiv:2010.03246*, 2020.

[5] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, 2017.

[6] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, 2018.

[7] D. Basu, D. Data, C. Karakus, and S. Diggavi. Qsparse-local-SGD: Distributed SGD with Quantization, Sparsification, and Local Computations. In *Advances in Neural Information Processing Systems*, 2019.

[8] A. Beznosikov, S. Horváth, P. Richtárik, and M. Safaryan. On biased compression for distributed learning. *arXiv preprint arXiv:2002.12410*, 2020.

[9] T. Boas, A. Dutta, X. Li, K. Mercier, and E. Niderman. Shrinkage function and its applications in matrix approximation. *The Electronic Journal of Linear Algebra*, 32:163–171, 2017.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[11] K. Bryan and T. Leise. Making do with less: An introduction to compressed sensing. *Siam Review*, 55(3):547–566, 2013.

[12] C-C Chang and C-J Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[13] W.-N. Chen, P. Kairouz, and A. Ozgur. Breaking the communication-privacy-accuracy trilemma. In *Advances in Neural Information Processing Systems*, 2020.

[14] T. Dettmers. 8-bit approximations for parallelism in deep learning. In *ICLR*, 2016.

[15] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[16] N. Dryden, T. Moon, S. A. Jacobs, and B. V. Essen. Communication quantization for data-parallel training of deep neural networks. *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, 2016.

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

[18] V. Gandikota, D. Kane, R. K. Maity, and A. Mazumdar. vqSGD: Vector quantized stochastic gradient descent. *arXiv preprint arXiv:1911.07971*, 2019.

[19] E. Gorbunov, D. Kovalev, D. Makarenko, and P. Richtarik. Linearly converging error compensated sgd. In *Advances in Neural Information Processing Systems*, 2020.

[20] J. Guo, W. Liu, W. Wang, J. Han, R. Li, Y. Lu, and S. Hu. Accelerating distributed deep learning by adaptive gradient quantization. In *ICASSP*, 2020.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *ICML*, 2015.

[22] V. Gupta, D. Choudhary, P. T. P. Tang, X. Wei, X. Wang, Y. Huang, A. Kejariwal, K. Ramchandran, and M. W. Mahoney. Fast distributed training of deep neural networks: Dynamic communication thresholding for model and data parallelism. *arXiv preprint arXiv:2010.08899*, 2020.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2015.

[24] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural Collaborative Filtering. In *WWW*, 2017.

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

[26] S. Horváth and P. Richtarik. A better alternative to error feedback for communication-efficient distributed learning. In *ICLR*, 2021.

[27] P. Jiang and G. Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In *Advances in Neural Information Processing Systems*, 2018.

[28] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *OSDI*, 2020.

[29] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *ICML*, 2019.

[30] D. P. Kingma and J. Ba. ADAM: A Method for Stochastic Optimization. In *ICLR*, 2015.

[31] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 1(4), 2009.

[32] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2017.

[33] Y. Lin, S. Han, H. Mao, Y. Wang, and W. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[35] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

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

[37] PyTorch. `https://pytorch.org/`.

[38] M. Safaryan, E. Shulgin, and P. Richtárik. Uncertainty principle for communication compression in distributed and federated learning and the search for an optimal compressor. *Information and Inference: A Journal of the IMA*, 2021.

[39] A. Sapio, M. Canini, C-Y Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik. Scaling Distributed Machine Learning with In-Network Aggregation. In *NSDI*, 2021.

[40] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *INTERSPEECH*, 2014.

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

[42] S. Shi, X. Chu, K. C. Cheung, and S. See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

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

[44] G. W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.

[45] S. U. Stich, J. B. Cordonnier, and M. Jaggi. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems*, 2018.

[46] S. U. Stich and S. P. Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication. *Journal of Machine Learning Research*, 2020.

[47] N. Strom. Scalable distributed DNN training using commodity GPU cloud computing. In *INTERSPEECH*, 2015.

[48] T. Vogels, S. P. Karimireddy, and M. Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*, 2019.

[49] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. ATOMO: Communication-efficient Learning via Atomic Sparsification. In *Advances in Neural Information Processing Systems*, 2018.

[50] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, 2018.

[51] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, 2017.

[52] C. Xie, S. Zheng, O. Koyejo, I. Gupta, M. Li, and H. Lin. Cser: Communication-efficient sgd with error reset. In *Advances in Neural Information Processing Systems*, 2020.

[53] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis. GRACE: A Compressed Communication Framework for Distributed Machine Learning. In *ICDCS'21*, Jul 2021.

[54] G. Yan, S-L Huang, T. Lan, and L. Song. DQSGD: Dynamic quantized stochastic gradient descent for communication-efficient distributed learning, 2021.

[55] S. Zheng, Z. Huang, and J. T Kwok. Communication-Efficient Distributed Blockwise Momentum SGD with Error-Feedback. In *Advances in Neural Information Processing Systems*, 2019.

[56] Y. Zhong, C. Xie, S. Zheng, and H. Lin. Compressed communication for distributed training: Adaptive methods and system. *arXiv preprint arXiv:2105.07829*, 2021.

[57] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2010.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] . See §3.1.
   (b) Did you include complete proofs of all theoretical results? [Yes] . All proofs are in §A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide code in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See §B.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Experimental setup in section §6.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See §B.1.
   (b) Did you mention the license of the assets? [Yes] We use open-source code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use open-source datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]