

FFT-based Gradient Sparsification for the Distributed Training of Deep Neural Networks

Linnan Wang

Brown University
Providence, RI, USA

Hang Liu

Stevens Institute of Technology
Hoboken, NJ, USA

Wei Wu

Los Alamos National Laboratory
Los Alamos, NM, USA

Junyu Zhang

University of Minnesota, Twin Cities
Minneapolis, MN, USA

George Bosilca

University of Tennessee
Knoxville, TN, USA

Maurice Herlihy

Brown University
Providence, RI, USA

Rodrigo Fonseca

Brown University
Providence, RI, USA

ABSTRACT

The performance and efficiency of distributed training of Deep Neural Networks (DNN) highly depend on the performance of gradient averaging among participating processes, a step bound by communication costs. There are two major approaches to reduce communication overhead: overlap communications with computations (lossless), or reduce communications (lossy). The lossless solution works well for linear neural architectures, e.g. VGG, AlexNet, but more recent networks such as ResNet and Inception limit the opportunity for such overlapping. Therefore, approaches that reduce the amount of data (lossy) become more suitable. In this paper, we present a novel, explainable lossy method that sparsifies gradients in the frequency domain, in addition to a new range-based float point representation to quantize and further compress gradients. These dynamic techniques strike a balance between compression ratio, accuracy, and computational overhead, and are optimized to maximize performance in heterogeneous environments.

Unlike existing works that strive for a higher compression ratio, we stress the robustness of our methods, and provide guidance to recover accuracy from failures. To achieve this, we prove how the FFT sparsification affects the convergence and accuracy, and show that our method is guaranteed to converge using a diminishing θ in training. Reducing θ can also be used to recover accuracy from the failure. Compared to STOA lossy methods, e.g., QSGD, TernGrad, and Top-k sparsification, our approach incurs less approximation error, thereby better in both the wall-time and accuracy. On an 8 GPUs, InfiniBand interconnected cluster, our techniques effectively accelerate AlexNet training up to 2.26x to the baseline of no compression, and 1.31x to QSGD, 1.25x to Terngrad and 1.47x to Top-K sparsification.

CCS CONCEPTS

- Computer systems organization → Neural networks; Heterogeneous (hybrid) systems;
- Theory of computation → Data compression; Machine learning theory.

KEYWORDS

Neural Networks, Machine Learning, FFT, Gradient Compression, Loosy Gradients

ACM Reference Format:

Linnan Wang, Wei Wu, Junyu Zhang, Hang Liu, George Bosilca, Maurice Herlihy, and Rodrigo Fonseca. 2020. FFT-based Gradient Sparsification for the Distributed Training of Deep Neural Networks. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20), June 23–26, 2020, Stockholm, Sweden*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3369583.3392681>

1 INTRODUCTION

Parameter Server (PS) and allreduce-style communications are two core parallelization strategies for distributed DNN training. In an iteration, each worker produces a gradient, and both parallelization strategies rely on the communication network to average the gradients across all workers. The gradient size of current DNNs is at the scale of 10^2 MB, and, even with the state-of-the-art networks such as Infiniband, repeatedly transferring such a large volume of messages over millions of iterations is prohibitively expensive. Furthermore, the tremendous improvement in GPU computing and memory speeds (e.g., the latest NVIDIA TESLA V100 GPU features a peak performance of 14 TFlops on single-precision and memory bandwidth of 900 GB/s with HBM2) further underscores communication as a bottleneck.

Recently, several methods have shown that training can be done with a lossy gradient due to the iterative nature of Stochastic Gradient Descent (SGD). It opens up new opportunities to alleviate the communication overhead by aggressively compressing gradients. One approach to compress the gradients is *quantization*. For example, Terngrad [29] maps a gradient into $[-1, 0, 1]$, and QSGD [4] stochastically quantizes gradients onto a uniformly discretized set larger than that of Terngrad. Such coarse approximation not only incurs large errors between the actual and quantized gradients as we demonstrate in Figure 15 [QSGD, TernGrad], but also fails to exploit

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

HPDC '20, June 23–26, 2020, Stockholm, Sweden

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7052-3/20/06...\$15.00

<https://doi.org/10.1145/3369583.3392681>

the bit efficiency in the quantization (Figure 7). Another approach to gradient compression, *sparsification*, only keeps the top-k largest gradients [2, 5, 14]. Similarly, Top-k loses a significant amount of actual gradients around zeros to achieve a high compression ratio (Figure 15, [Top-k]). In summary, existing lossy methods greatly drop gradients, incur large approximation errors (Figure 15e), leading to the deterioration of the final accuracy (Table 2). To avoid compromising the convergence speed, both *quantization* and *sparsification* must limit the compression ratio, leading to sub-optimal improvement of the end-to-end training wall time.

In this paper, we propose a gradient compression framework that takes advantages of both *sparsification* and *quantization* with two novel components, FFT-based sparsification, and a range-based quantization. FFT-based sparsification allows removing the redundant information, while preserving the most relevant information (Figure 15 [FFT]). As a result, FFT incurs fewer errors in approximating the actual gradients (Figure 15e), thereby better in accuracy than QSGD, TernGrad, and Top-K (Table 2). We treat the gradient as a 1D signal, and drop near-zero coefficients in the frequency domain, after an FFT. Deleting some frequency components after the FFT introduces magnitude errors, but the signal maintains its distribution (Figure 5). As a result, the sparsification in the frequency domain can achieve the same compression ratio as in the spatial domain but preserving more relevant information.

To further improve the end-to-end training wall time, we introduce a new range-based variable precision floating point representation to quantize and compress the gradient frequencies after sparsification. Most importantly, unlike the uniform quantization used in existing approaches, the precision of representable floats in our method can be adjusted to follow the distribution of the original gradients (Figure 9). The novel range-based design allows us to fully exploit the precision given limited bits so that the approximation error can be further reduced. By combining *sparsification* and *quantization*, our framework delivers a higher compression ratio than the single method, resulting in shorter end-to-end training wall time than QSGD, Terngrad, and Top-k.

Lastly, our compression framework is highly efficient and scalable. The primitive algorithms in our compression scheme, such as FFT, top-k select, and precision conversions, are efficiently parallelizable and thus GPU-friendly. We resort to existing highly optimized GPU libraries such as cuFFT, Thrust, and bucketSelect [3], while we propose a simple yet efficient packing algorithm to transform sparse gradients into a dense representation. Minimizing the computational cost of the compression is crucial for high-speed networks, such as Infiniband networks, as we analyzed in Figure 10.

Specifically, the contributions of this paper are as follows:

- a novel FFT-based, tunable gradient sparsification that retains the original gradient distribution.
- a novel range-based variable precision floating-point that allocates precision according to the gradient distribution.
- a analytic model to guide people when to enable compression and how to set a compression ratio according to hardware specifications.
- the convergence proof of our methods, and its guidance in selecting a compression ratio θ , to ensure the convergence, or reduce θ to recover the accuracy. To the best of our knowledge,

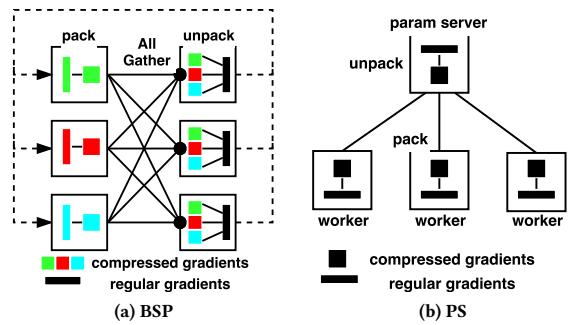


Figure 1: Two parallelization schemes of distributed DNN training:(a) Bulk Synchronous Parallel (BSP) strictly synchronizes gradients with all-to-all group communications, e.g. MPI collectives; (b) Parameter Server (PS) exchanges gradients with point-to-point communications, e.g. push/pull.

this paper is the first one to discuss the relationship between compression ratio and accuracy of neural networks.

- highly optimized system components for a compression framework that achieves high throughput on GPUs and is beneficial even on state-of-the-art Infiniband networks.

2 BACKGROUND AND MOTIVATION

In general, there are two strategies to parallelize DNN training: *Model Parallelism* and *Data Parallelism*. *Model Parallelism* splits a network into several parts, with each being assigned to a computing node [10]. It demands extensive intra-DNN communications in addition to gradient exchanges. It largely restricts the training performance, and thereby *Model Parallelism* is often applied in scenarios where the DNN cannot fit onto a computing node [10]. The second approach, *Data Parallelism* [26], partitions the image batch, and every computing node holds a replica of the network. In a training iteration, a node computes a sub-gradient with a batch partition. Then, nodes *all-reduce* sub-gradients to reconstruct the global one. The only communications are for necessary gradient exchanges. Therefore, current Deep Learning (DL) frameworks such as SuperNeurons [27], MXNet [8], Caffe [16], and TensorFlow [1] parallelize the training with *Data Parallelism* for the high-performance.

There are two common strategies to organize the communications with data parallelism: with a centralized Parameter Server (PS) (Figure 1b), or with all-to-all group communications, e.g., *allreduce* (Figure 1a). TensorFlow [1], MXNet [8], and PaddlePaddle implement distributed DNN training with a Parameter Server (PS) [19]. In this distributed framework, the parameter server centralizes the parameter updates, while workers focus on computing gradients. Each worker pushes newly computed gradients to the parameter server, and the parameter server updates parameters before sending the latest parameters back to workers. Though this client-server [7] style design easily supports fault tolerance and elastic scalability, the major downside is the network congestion on the server. Alternatively, *allreduce*-based Bulk Synchronous Parallel SGD can better exploit the bandwidth of a high-speed, dense interconnects, such as modern Infiniband networks. Instead of using a star topology, *allreduce* pipelines the message exchanges at a fine granularity with

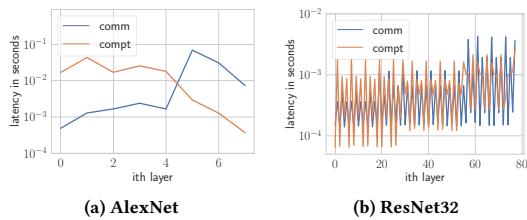


Figure 2: layer-wise communications (all-reduce) v.s. computations in an iteration of BSP SGD using 16 P100 (4 GPUs/node with 56Gbps FDR).

adjacent neighbors in a ring-based topology. Since the pipeline fully utilizes the inbound and outbound link of every computing node, it maximizes network bandwidth utilization and achieves appealing scalability where the cost is largely independent of the number of computing nodes.

There are trade-offs between the BSP and PS schemes, with PS having better fault tolerance, and *allreduce* better exploits the network bandwidth. However, as we argue below, in both cases, the communication cost is high, and reducing it can yield substantial gains in training latency.

2.1 Communication Challenges in Distributed Training of DNNs

Communications for averaging sub-gradients is widely recognized as a major bottleneck in scaling DNN training[10, 26, 30]. With increasing data complexity and volume, and with emerging non-linear neural architectures, two critical issues exacerbate the impact of communications in the scalability and efficiency of distributed DNN training with data parallelism: I) *the increasing amounts of data to be exchanged*, and II) *the decreasing opportunity to overlap computation and communication*.

Challenge I: Enormous amounts of communications during training. DNNs are extremely effective at modeling complex non-linearities thanks to the representation power of millions of parameters. The number of parameters dictates the size of the gradients. Specifically, the gradient sizes of AlexNet, VGG16, ResNet32, and Inception-V4 are 250MB, 553MB, 102MB, and 170MB. Even with the highly optimized allreduce implementation on a 56 Gbps FDR network, communication overhead remains significant. For example, the communication for AlexNet, VGG16, Inception-V4 and ResNet32 at regular single-GPU batch sizes¹ consumes 64.17%, 18.62%, 33.07% and 43.96% of an iteration time, respectively.

Challenge II: Decreasing opportunity to overlap computation and communication. One promising solution to alleviate the communication overhead is hiding the communication for averaging the gradient of the i^{th} layer by the computation of $i-1^{th}$ layer in the backward pass. This lossless technique has proven to be effective on linear networks such as AlexNet and VGG16 [6, 23], as these networks utilize large convolution kernels to process input data. Figure 2a demonstrates the computation time of the convolution layers is 10× larger than the communication time, easy for overlapping. However, the overlapping technique is not always applicable

¹the single GPU batch size for AlexNet is 64, and 16 for others.

for two reasons. *First*, the degree of overlapping is largely decided by the computation pattern of the neural network model. The opportunity for computation and communication overlap is very limited in recent neural architectures, such as Inception-V4 [25] and ResNet [15]. The sparse fan-out connections in the Inception Unit (Figure 1a in [27]) replace one large convolution (e.g. 11×11 convolution kernel in AlexNet) with several small convolutions (e.g. 3×3 convolution kernels). Similarly, ResNet utilizes either 1×1 or 3×3 small convolution kernels. As a result, the layer-wise computational cost of ResNet is similar to or smaller than communication (Figure 2b); hence, it is much harder to overlap these neural networks than AlexNet. *Second*, the degree of overlapping is also impacted by the bandwidth of networks. With slower networks, there are less opportunity to overlap communications and computations. Specifically, as seen in Figure 2a, the computation cost of convolution layers of the AlexNet is 10× larger than the communication cost with 56Gbps InfiniBand. However, when training AlexNet in a low profile network such as 1Gbps Ethernet, it becomes impossible to hide the communication cost as it is significantly larger than the computation cost.

These two challenges – increasing data exchanged, and decreasing opportunity to hide communication latency – make it attractive to look for solutions that minimize the communication cost by decreasing the communication volume. Training a neural network with imprecise gradient updates still works as parameters are iteratively refined [2]. Particularly, lossy gradient compression can achieve higher compression rates and still allow the network to deliver target accuracy [4]. Given this, it is not surprising that several gradient compression approaches have been proposed in the literature. They generally fall into two categories: quantization of the gradients (e.g. [4, 9, 24, 29]), where these are represented with lower precision numbers, and sparsification (e.g. [2, 5, 28]), where small gradient are treated as zero and not transmitted. We discuss these approaches in detail in Section 5. As we describe next, we propose a novel gradient compression scheme that uses adaptive quantization and tunable FFT-based gradient compression that, together, achieve variable compression ratios that can maintain convergence quality, and, critically, is cheap enough computationally to be beneficial.

3 METHODOLOGY

3.1 The Compression Framework

Figure 3 provides a step-by-step illustration of our compression pipeline.

- 1 Linearize the gradients by re-arranging gradient tensors into a 1-d vector for Fast Fourier Transform (FFT), which is discussed in Section 3.1.1.
- 2 Truncate the gradient frequencies based on their magnitudes to sift out the **top-k** low-energy frequency components, which is discussed in Section 3.1.1.
- 3 Transform the frequencies’ representation from 32-bit float to a new, **range-based**, N -bit float ($N < 32$) to further compress down the gradient frequency, which is discussed in Section 3.2.1.
- 4 **Pack** sparse data into dense vector and transfer them out, which is discussed in Section 3.1.1.

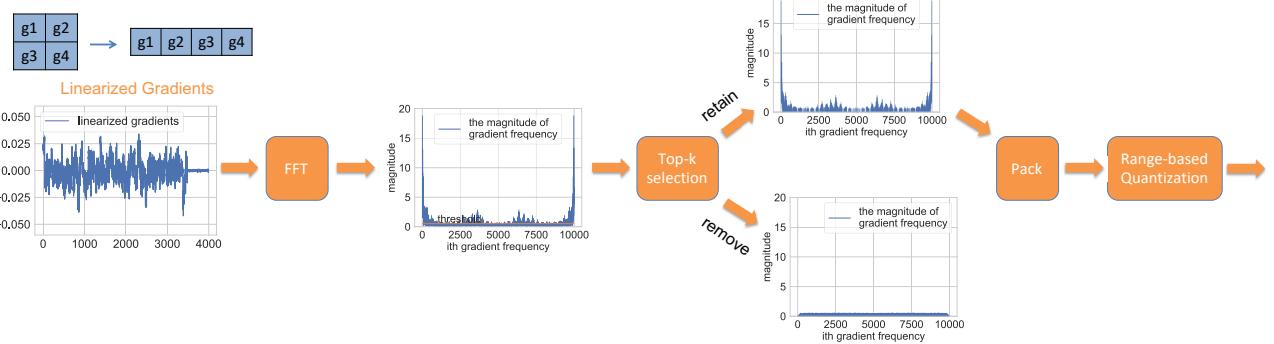


Figure 3: The gradient compression framework (sender).

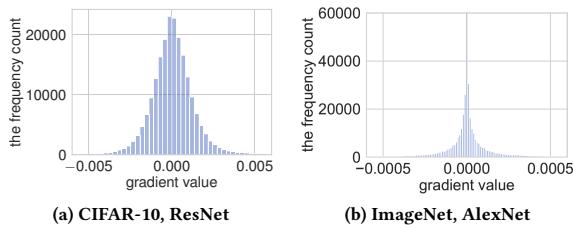
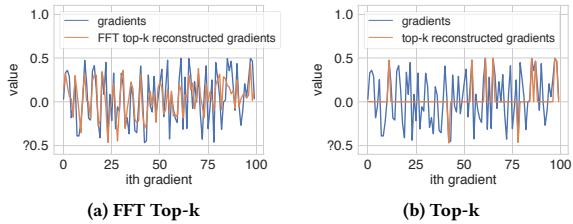
Figure 4: Histogram of DNN gradients: we sampled gradients every 10^3 and 10^4 iterations in a full training.

Figure 5: FFT Top-k v.s. direct Top-k sparsification: Top-k aggressively loses gradients (err=0.0246), while FFT preserves more relevant information (err=0.0209) at the same sparsification ratio.

On the receiver side, a similar approach (but using the inverse operations in the reverse order) is used to decompress the gradient frequency vector into gradients. Detailed discussions of compression components and their motivations are as follows.

3.1.1 Removing redundant information with FFT based Top-K sparsification. **Motivation:** the gradient points to a descent direction in the high dimensional space, thereby small perturbations on gradients can be viewed as introducing local deviations along the descent direction. If such deviations are limited during the training, these imprecise descent directions still iteratively lead to a local optimum at the cost of additional iterations. This is the intuition for the gradient sparsification. Besides, Figure 4 indicates high redundancy in DNN gradients due to a lot of near-zero components, that may have limited contributions in updating gradients. Recently, several top-k based methods [2, 5, 14] have also shown the possibility to train DNNs with only the top 10% largest gradients. However, the

resulting gradients, as shown in Figure 5, significantly deviate from the original, for entirely dropping the gradients below the threshold. This has motivated us to sparsify gradients, instead, in the frequency domain for preserving the trend of the original signal even after removing the same amount of information. For a gradient vector of length N , each gradients is $g_i = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi kn}{N}}$ after FFT. If we sparsify on x_n , i.e. $g_i = \sum_{n=0}^{top_k} x_n e^{-\frac{i2\pi kn}{N}}$, g_i still preserves some of the original gradient information. Therefore, FFT based top-k shows better results than top-k in Figure 5. More validations are available in the experimental section.

Our approach: The detailed computation steps of our FFT sparsification are highlighted in Figure 3. Recent generations of NVIDIA GPUs support mixed-precision; and computing with half-precision increases the FFT throughput up to $2\times$. So, we convert 32-bit (full-precision) gradients into 16-bit (half-precision) gradients to improve the throughput before applying FFT, and the information loss from the conversion is negligible due to the bounded gradients.

After FFT, the next step is to filter the low energy gradient in the frequency domain. We introduce a new hyper-parameter, θ , to regulate the sparsity of frequencies. Here, we only describe the procedures, and the tuning of θ is thoroughly discussed in Section 3.5 and experiments. If $\theta = 0.9$, we keep the top 10% frequency components in magnitude and drop the rest by resetting to zeros (Figure 3). The selection is implemented with either sorting or Top-k. Since Thrust² and cuFFT³ provide highly optimized FFT and sorting kernels for the GPU architecture, we adopted them in our implementations.

3.2 Packing sparse data into a dense vector

Thresholding gradient frequencies in the last step yields a highly irregular sparse vector, and we need to pack it into a dense vector to reduce communications. The speed of packing a sparse vector is critical to the practical performance gain. Here, we propose a simple parallel packing algorithm:

- 1 Create a *status* vector and mark an element in *status* as 1 if the corresponding scalar in *sparse* vector is non-zero (e.g., *sparse* = $[a, 0, b, 0, c, 0, 0]$ and *status* = $[1, 0, 1, 0, 1, 0, 0]$).

²<https://developer.nvidia.com/thrust>

³<https://developer.nvidia.com/cufft>

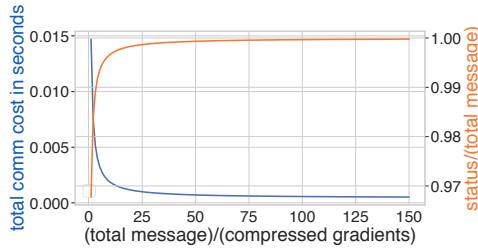


Figure 6: the effect of status vector: given 100 MB gradients, the improvement after dropping > 95% gradients ($\theta = 0.05$, compression ration is 20) is limited.

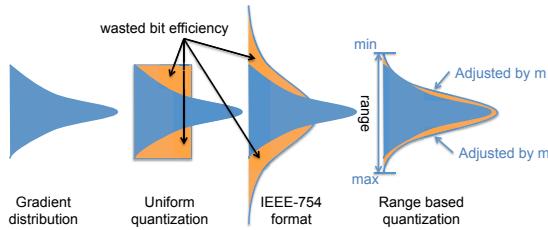


Figure 7: comparisons of quantization schemes: the uniform distribution and IEEE 754 format.

- 2 Perform a parallel prefix-sum on *status* to generate a *location* vector ($[1, 1, 2, 2, 3, 3, 3]$).
- 3 if $status[i] == 1$, write *sparse*[*i*] to *dense*[*location*[*i*]], and *dense* vector is the packed result.

This parallel algorithm has a 689× speedup over the single-threaded algorithm on a TESLA V100 with a throughput of 34 GB/s.

We need to send the status vector and the compressed gradient to perform the decompression. The status vector is a bitmap that tracks the location of non-zero elements, and its length in bits is the same as the gradient vector. Figure 6 shows the cost of the status vector is non-negligible after the compression ratio exceeding 20. Therefore, setting $\theta < 0.05$ is not desired.

3.2.1 Range based Quantization. Motivation: the range of single precision IEEE-754 floating point is $[-3.4 \times 10^{38}, +3.4 \times 10^{38}]$, while the range of gradients and their frequencies are much smaller (e.g. $[-1, +1]$). This motivates us to represent the bounded gradients with fewer bits. The problem of using an N bits IEEE 754 format, as seen in Figure 7, is the inconsistency between the range of gradients $[min, max]$ and the range of the IEEE representable numbers. Given N bits for IEEE 754, there are $N - 2$ combinations of exponent-mantissa. The representation range is either too large or too small for gradients, regardless of which combinations to choose. Another conventional way is to equally divide the $max - min$ into 2^N , i.e., uniform quantization. Still, the actual gradient distribution is far from the uniform, and thereby it is also inefficient, as shown in Figure 7.

Our approach: we propose an offset-based N-bit floating point, which intends to match the distribution of representable numbers to the real gradients. Our representation is to use the N-bit binary format of a positive number as base number *pbase*, and encode it to 0...01. The rest positive numbers are encoded as 0...01 (*pbase*) +

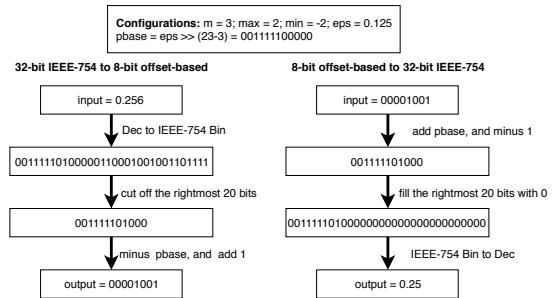


Figure 8: Illustration of range based quantizer: an example conversion of between 32 bits IEEE 754 and 8 bits our representation.

offset. The negative numbers also follow the same rule. Therefore, the total 2^N representable numbers consist of P positive numbers and $2^N - P$ negative numbers. To match the range of real gradients, our quantization permits the manual setting of a representation range, defined by *min* and *max*. We estimate *min* and *max* from the first few iterations of gradients. Then, we tune *m* and *eps* to adjust the precision of representable numbers, as shown in Figure 7. *m* represents the number of bits left for the mantissa, and *eps* represents the minimal representable positive number whose corresponding N-bit binary is *pbase*. The following further explains how *m* and *eps* adjust the precision:

- *m*: let's denote the difference between two consecutive numbers as *diff*. For *m* bits mantissa, the exponent increases by 1 after 2^m number, and increasing *diff* = *diff* * 2. Since *diff* is exponentially growing, this creates a Gaussian like representation range that matches to real gradients. If *max*, *min* and *eps* are fixed, *P* is small for a small *m*, as it takes fewer numbers to increase the exponent. Similarly, a large *m* leads to a larger *P*. Therefore, *m* is very sensitive for precision.
- *eps*: with *max*, *min* and *m*, *diff* is also fixed. If *eps* is small, it takes more steps to reach *max* yielding a large *P*; and vice versa.

Since *m* and *eps* determine *P*, we need to tune them to make *P* close to $2^N/2$ for balancing the range of positive and negative numbers. In practice, *N*, *min*, and *max* are empirically decided from gradients, and the *m* $\in [1, N]$. We iterate every *m* to tune for *eps*. Given *N*, *m*, *min*, and *max*, we initialize *eps* as a reasonably small number, e.g., 0.002, then de-compress the 1..1 (the minimal representable negative number) back to FP32 with the selected *eps*, and the resulting number is the current actual minimal negative number *actual_min*; if *actual_min* is smaller than *min*, we decrease *eps*, and increase otherwise. Following this path, *P* converges to $2^N/2$, a state with equal positive and negative numbers, and yielding the optimal *eps*.

Alg. 1 summarizes the conversion from 32-bit IEEE 754 to our N-bit offset based float, and *N* is set w.r.t the precision requirement for the training. Figure 8 provides a step-by-step conversion between IEEE 754 and our 8 bits representation.

Figure 9 shows the resulting number distributions of our approach when the range is set to $[-0.5, 0.5]$, and $[-5, 5]$. This shows our approach successfully adjusts representation ranges, while still

Algorithm 1: Offset-based N -bit floating point

```

Input: init(min, max)
1 pbase_binary = eps » (23-m);
Input: 32bit_to_Nbit(32bit_float)
2 if 32bit_float > max then
3   32bit_float = max;
4 32bit_binary = 32bit_float » (23-m);
5 Nbit_binary = 32bit_binary · pbase_binary + 1;
Input: Nbit_to_32bit(Nbit_binary)
6 32bit_binary = Nbit_binary + pbase_binary - 1;
7 32bit_float = 32bit_binary « (23-m);

```

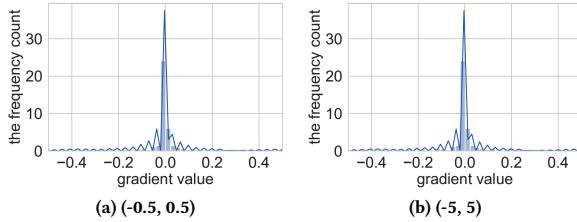


Figure 9: Adjustable representation range: our quantization successfully adjusts its distribution.

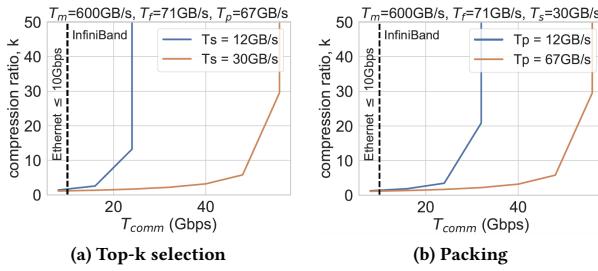


Figure 10: Minimal compression ratio k exhibits performance benefits at different network bandwidths T_{comm} , packing throughput T_p and selection throughput T_s . It is easy to get performance improvement from a slow network, while it requires faster compression primitives to be beneficial on a fast network.

maintaining similar distribution to actual gradients. This is because $diff$ increases $2x$ after 2^m numbers, leading to more numbers around 0, and less to max or min . Unlike prior static approach, our offset based float dynamically changes the representable range to sustain the various precision requirements from different training tasks. Besides, the float quantizations are embarrassingly data-parallel, so it is easy to achieve the high-performance.

3.3 Sensitivity Analysis

The compression cost shall not offset the compression benefit to acquire practical performance gain. In this section, we analyze the performance of compression primitives and their impact on perceived network bandwidth. Table 1 defines all symbols used in the analysis. It is noted that we use the same notation T_m for both float-to-half and range-based quantization as they are $O(N)$ algorithms and embarrassingly parallel. Given a message of size

Symbol	Explanation
T_m	Maximum throughput of precision conversion including float-to-half and range-based quantization
T_f	Maximum throughput of FFT
T_p	Maximum throughput of packing
T_s	Maximum throughput of top-k selection
T_{comm}	Maximum throughput of communication via networks
k	Overall compression ratio

Table 1: Symbols of equations in Section 3.3.

M , the cost of compression is:

$$cost_{comp} = M \left(\frac{2}{T_m} + \frac{1}{T_f} + \frac{1}{T_p} + \frac{1}{T_s} \right) \quad (1)$$

The communication cost after compression is :

$$cost_{comm} = \frac{M}{T_{comm}} \left(\frac{1}{k} \right) \quad (2)$$

So the communication cost saved by compression is:

$$saved_cost_{comm} = \frac{M}{T_{comm}} \left(1 - \frac{1}{k} \right) \quad (3)$$

To compensate for the cost of compression and decompression, $2cost_{comp} < saved_cost_{comm}$ must hold to acquire the practical performance gain, that is

$$k > \frac{1}{1 - 2T_{comm} \left(\frac{2}{T_m} + \frac{1}{T_f} + \frac{1}{T_p} + \frac{1}{T_s} \right)} \quad (4)$$

The performance of T_m depends on the hardware characteristics (such as GPU DRAM bandwidth), and T_f depends on cuFFT. It is therefore reasonable to consider them fixed for a particular GPU hardware. T_s and T_p depend on the libraries and algorithms applied. By varying T_s and T_p in Equation 4 we analyze the minimal compression ratio k that will show benefits for a particular network infrastructure. Figure 10 shows the relationship between k and T_{comm} . If the network throughput is low, like Ethernet, a small k could compensate for the cost of compression and decompression, which means increasing k would significantly boost the performance of communications. For example, Figure 10 shows that $k = 2$ is enough to compensate for the overhead of compression and decompression on a 10Gbps Ethernet. On the other hand, if the network throughput is high, like InfiniBand, a larger k would be necessary; otherwise, the overall performance will be impacted by the overhead of compression and decompression. More precisely, the red line in Figure 10 indicates that the minimal compression ratio k should be about 30 to exhibit any benefit on a 56Gbps InfiniBand.

Figure 10 also predicts that the performance of the compression primitives is crucial for high bandwidth networks. As seen in Figure 10.a, if T_s is 12GB/s, for any T_{comm} larger than 22Gbps no compression ratio will be able to provide any tangible communication improvement.

3.4 Convergence Analysis

In order to analyse the convergence of our proposed technique we formulate the DNN training as:

$$\min_x f(x) := \frac{1}{N} \sum_{i=1}^N f_i(x), \quad (5)$$

where f_i is the loss of one data sample to a network. For non-convex optimization, it is sufficient to prove the convergence by showing $\|\nabla f(x^t)\|^2 \leq \epsilon$ as $t \rightarrow \infty$, where ϵ is a small constant and t is the iteration. The condition indicates the function converges to the neighborhood of a stationary point. Before stating the theorem, we need to introduce the notion of Lipschitz continuity. $f(x)$ is smooth and non-convex, and ∇f are L -Lipschitz continuous. Namely,

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

For any x, y ,

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|x - y\|^2.$$

ASSUMPTION 3.1. Suppose j is a uniform random sample from $\{1, \dots, N\}$, then we make the following bounded variance assumption:

$$\mathbb{E}[\|\nabla f_j(x) - \nabla f(x)\|^2] \leq \sigma^2, \text{ for any } x.$$

This is a standard assumption widely adopted in the SGD convergence proof [21] [13]. It holds if the gradient is bounded.

ASSUMPTION 3.2. In the data-parallel training, the gradient of each iteration is $\bar{v} = \frac{1}{p} \sum_1^p v_i$; p is the number of processes, and v_i is the gradient from the i^{th} process. Let's denote $\theta \in [0, 1]$ to control the percentage of information loss in the compression function $\hat{v}_i = T(v_i, \theta)$ that does $\text{quant}(\text{FFT-sparsification}(v_i))$, so $\tilde{v} = \sum_1^p \hat{v}_i$. We assume there exists a α such that:

$$\|\bar{v} - \tilde{v}\| \leq \alpha \|\bar{v}\|.$$

So, \hat{v} only loses a small amount of information with respect to \bar{v} , and the update from the sparsified gradient is within a bounded error range of true gradient update. It is a necessary condition for deriving the upper bound.

With our compression techniques, one SGD update becomes:

$$x^{t+1} = x^t - \eta_t \left(\frac{1}{p} \sum_1^p \hat{v}_i \right) = x^t - \eta_t \tilde{v}_t. \quad (6)$$

Then, we have the following lemma for one step:

LEMMA 3.3. Assume $\eta_t \leq \frac{1}{4L}, \theta_t^2 \leq \frac{1}{4}$. Then

$$\frac{\eta_t}{4} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \mathbb{E}[f(x^t)] - \mathbb{E}[f(x^{t+1})] + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t}. \quad (7)$$

Please check the supplemental material for the proof of this lemma. Summing over (7) for K iterations, we get:

$$\sum_{t=0}^{K-1} \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2] \leq 4(f(x^0) - f(x^K)) + \sum_{t=1}^{K-1} (L\eta_t + \theta_t^2) \frac{2\eta_t \sigma^2}{b_t}. \quad (8)$$

Next, we present the convergence theorem.

THEOREM 3.4. If we choose a fixed learning rate, $\eta_t = \eta$; a fixed dropout ratio in the sparsification function, $\theta_t = \theta$; and a fixed mini-batch size, $b_t = b$; then the following holds:

$$\min_{0 \leq t \leq K-1} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \frac{4(f(x^0) - f(x^{K-1}))}{K} + (L\eta + \theta^2) \frac{2\eta \sigma^2}{b}.$$

Proof. $\min_{0 \leq t \leq K-1} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \frac{1}{K} \sum_{t=0}^{K-1} \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2]$, as $\|\nabla f(x^t)\|^2 \geq 0$. By (8), we get the theorem. \square

THEOREM 3.5. If we apply the diminishing stepsize, η_t , satisfying $\sum_{t=0}^{\infty} \eta_t = \infty, \sum_{t=0}^{\infty} \eta_t^2 < \infty$, our compression algorithm guarantees convergence with a diminishing drop-out ratio, θ_t , if $\theta_t^2 = L\eta_t$.

Proof. If we randomly choose the output, x_{out} , from $\{x^0, \dots, x^{K-1}\}$, with probability $\frac{\eta_t}{\sum_{t=0}^{K-1} \eta_t}$ for x^t , then we have:

$$\mathbb{E}[\|\nabla f(x_{out})\|^2] = \frac{\sum_{t=0}^{K-1} \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2]}{\sum_{t=0}^{K-1} \eta_t} \quad (9)$$

$$\leq \frac{4(f(x^0) - f(x^*))}{\sum_{t=0}^{K-1} \eta_t} + \frac{\sum_{t=0}^{K-1} (L\eta_t + \theta_t^2) 2\eta_t \sigma^2}{b \sum_{t=0}^{K-1} \eta_t}. \quad (10)$$

Note that $\sum_{t=0}^{K-1} \eta_t \rightarrow \infty$, while $\sum_{t=0}^{K-1} (L\eta_t + \theta_t^2) 2\eta_t \sigma^2 = \sum_{t=0}^{K-1} 4L\eta_t^2 \sigma^2 < \infty$, and we have $\mathbb{E}[\|\nabla f(x_{out})\|^2] \rightarrow 0$. \square

4 EVALUATION

Our experiments consist of two parts to assess the proposed techniques. First, we validate the convergence theory and its assumptions with AlexNet on ImageNet and ResNet32 on CIFAR10, which sufficiently cover typical workloads in traditional linear and recent non-linear neural architectures, and also provide coverage on two widely used datasets. Then, we show that the FFT-based method demonstrates better convergence and faster compression than other state-of-the-art compression methods such as QSGD [4], TernGrad [29], Top-k sparsification [5, 20], as our techniques incur fewer approximation errors, while still delivering a competitive compression ratio for using both sparsification and quantization.

Parallelization scheme: we choose BSP for parallelization for its simplicity in the theoretical analysis: BSP follows strict synchronizations, allowing us to better observe the effects of gradient compression toward the convergence by iterations.

Implementation: we implemented our approach, losses SGD(no compression), QSGD, Top-K, and TernGrad in a C++ DL framework, SuperNeurons [27]; We used the allgather collective from NVIDIA NCCL2 to exchange compressed gradients since existing communication libraries lack the support for sparse all-reduce (Figure 1a). Even though SGD usually uses allreduce instead of allgather as it does not have compression; for a fair comparison, we applied allgather for all algorithms to demonstrate the algorithmic benefit of our FFT compression. Every GPU has a copy of global gradients for updating parameters after all-gather local gradients. Parameters need to be synchronized after multiple iterations to eliminate the precision errors, and here we broadcast parameters every 10 iterations. It is noticed that we did not adopt communication and computation overlapping strategy as it could be another optimization method orthogonal to compression, and is not in the scope of this paper.

Training setup: The single GPU batch is set to 128 and 64 for ResNet32 and AlexNet, respectively. The momentum for both networks is set to 0.9. The learning rate for Resnet32 is 0.01 at epochs $\in [0, 130]$, and 0.001 afterwards; the learning rate for AlexNet is 0.01 at epochs $\in [0, 30]$, 0.001 at epochs $\in [30, 60]$, and 0.0001 afterwards.

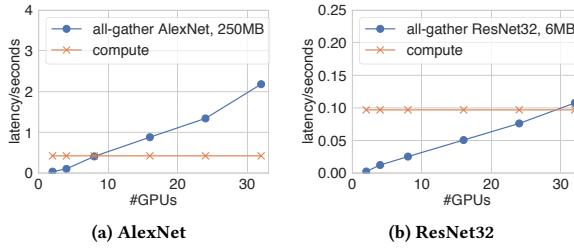


Figure 11: the latency for *all-gather* AlexNet and ResNet32 from 2 to 32 GPUs.

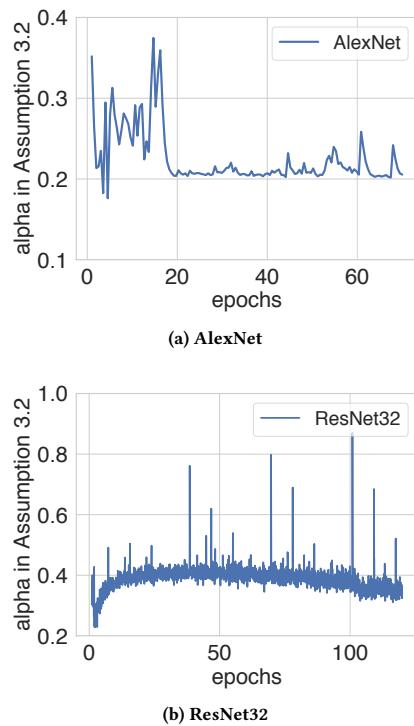


Figure 12: Empirical verification of Assumption 3.2.

Machine setup: we conducted experiments on the Comet cluster hosted at San Diego Supercomputer Center. Comet has 36 GPU nodes, and each node is equipped with 4 NVIDIA TESLA P100 GPUs and 56 Gbps FDR InfiniBand, Figure 11 shows the *allgather* cost almost linearly increases with the number of GPUs. This is because the total exchanged messages in *allgather* linearly increase with #GPUs [12]. In our experiments, we used 8 GPUs in evaluating the accuracy and performance when integrating our compression methods in training, and up to 32 GPUs in evaluating the scalability of the distributed training.

4.1 Validation of Theorems

Verification of assumptions: our convergence theorems rely on Assumption 3.1 and Assumption 3.2. Assumption 3.1 automatically holds due to the bounded gradients. Assumption 3.2 always hold

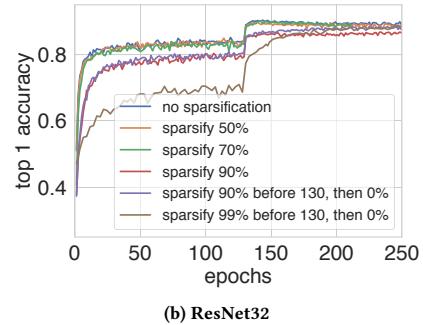
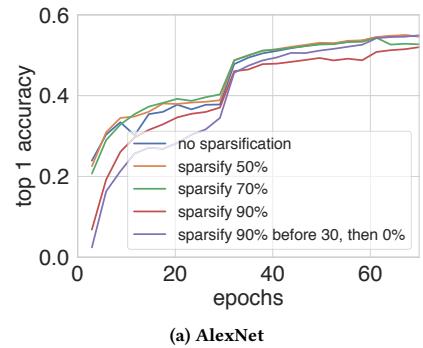


Figure 13: Empirical validation of Theorem 3.5.

if $p = 1$, but it can break in very rare cases for $p > 1$. For example, α does not exist if $\bar{v} = [0, 0]$, given two opposite gradients, e.g. $\bar{v}_1 = [-0.3, 0.5]$ and $\bar{v}_2 = [0.3, -0.5]$. Though the scenario is very unlikely, we empirically validate Assumption 3.2 on different training tasks by calculating $\alpha = \frac{\|\bar{v} - \hat{v}\|}{\|\bar{v}\|}$. From Figure 12, $\alpha \in [0, 1]$ practically sustaining Assumption 3.2.

Validation of theorems: Theorem 3.4 states a large compression ratio, i.e. large θ , can jeopardize the convergence, and theorem 3.5 states that our FFT-based sparsified SGD is guaranteed to converge with a diminishing compression ratio. The goal of optimization is to find a local optimum, where the gradient approximates to zero, i.e. $\mathbb{E}[\|\nabla f(x^t)\|^2] \rightarrow 0$, as $K \rightarrow \infty$. From the inequality in theorem 3.4, $\frac{4(f(x^0) - f(x^{K-1}))}{K} \rightarrow 0$ as $K \rightarrow \infty$, leaving $E[\|\nabla f(x^t)\|^2]$ bounded by $(L\eta + \theta^2) \frac{2\eta\sigma^2}{b}$. $L\eta \frac{2\eta\sigma^2}{b}$ is the error term from SGD, and $\theta^2 \frac{2\eta\sigma^2}{b}$ is the error term from the compression. Compared to the SGD, using a large θ in the gradient compression slacks off the bound for $\mathbb{E}[\|\nabla f(x^t)\|^2]$, causing the deterioration on both the validation accuracy and training loss. As shown in Figure 13, when $\theta = 0.5$ (i.e., sparsify 50%), the accuracy and loss traces of AlexNet and ResNet32 behave exactly the same as SGD (shown as no sparsification). When $\theta = 0.9$ (i.e., sparsify 90%), both the training loss and validation accuracy significantly deviate from SGD, as a large θ increases the error term $\frac{2\eta\sigma^2\theta^2}{b}$ loosening the bound for $\mathbb{E}[\|\nabla f(x^t)\|^2]$. Therefore, θ is critical to retain the same performance as SGD, and it is tricky to select θ in practice. We present Theorem 3.5 to resolve this issue. The theorem compensates for Theorem 3.4, indicating that a large θ can still deliver the same accuracy as SGD if we shrink the θ during the training.

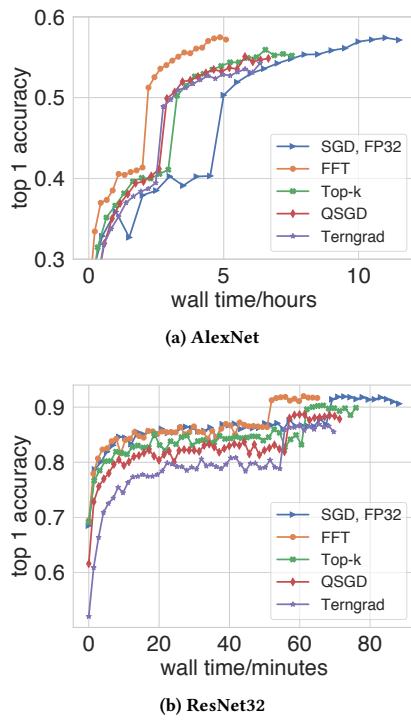


Figure 14: Training wall time on a 8 GPUs cluster: FFT outperforms TernGrad, QSGD and Top-k in both the speed and test accuracy. FFT is faster for a high compression ratio by combining sparsification and quantization, while the better gradient quality of FFT explains the good accuracy, as we will show in Figure 15.

Empirical results in Figure 13 validate Theorem 3.5. For example, by setting $\theta = 0.9$ (drop 90%, red line), both AlexNet and ResNet32 fail to converge to the same case of SGD. However, it is able to bring the accuracy back to the same result as the SGD in the same epochs simply by diminishing θ from 0.9 to 0 at the 30th epoch for AlexNet, and at the 130th epoch for ResNet32. Therefore, we claim both Theorem 3.4 and Theorem 3.5 are legitimate.

Implications of theorems: these two theorems explain the relationship between the accuracy and compression ratio θ , and act as a guide to help preserve the training network accuracy by tuning the compression ratio during the training. Hence, in practice, to ensure the convergence, we can shrink θ along with the learning rate η for the condition of $\theta_t^2 = L\eta_t$. In order to recover the accuracy, we can also reduce θ as the case in Fig. 13 that a failure case ($\theta = 0.9$) recovers the accuracy after reducing θ to 0 in the middle of training.

4.2 Algorithm Comparisons

Choice of Algorithms: Here we evaluate our FFT-based techniques against 3 major gradient compression algorithms, Top-k sparsification [2, 5, 20], and Terngrad [29] and QSGD [4]. The baseline method is SGD using 32 bits float. Top-k sparsification thresholds the gradients w.r.t their magnitude, and the compression ratio is determined by $1/(1-\theta)$, where θ is the drop-out ratio. Please note that Top-k

Method	AlexNet top1 acc	Speedup w.r.t SGD	ResNet32 top1 acc	Speedup w.r.t SGD
SGD, FP32	56.52%	1	92.11%	1
FFT	56.61%, (+0.09%)	2.26	91.99%, (-0.12%)	1.33x
Top-K	55.07%, (-1.45%)	1.53	90.31%, (-1.80%)	1.12x
QSGD	53.54%, (-2.98%)	1.73	88.66%, (-3.45%)	1.21x
TernGrad-noclip	52.86%, (-3.66%)	1.81	86.90%, (-5.21%)	1.24x

Table 2: Summarization of Figure 14: the difference of test accuracy and the speedup over lossless SGD.

variant e.g. DGC [20] utilizes heuristics like error accumulation and momentum correction to boost performance. To fairly evaluate Top-k sparsification against FFT based sparsification, we evaluated the vanilla Top-k v.s. the vanilla FFT sparsification, and finding heuristics to boost FFT sparsification is orthogonal to this study. Both Terngrad and QSGD map gradients to a discrete set. Specifically, Terngrad maps each gradient to the set of $\{-1, 0, 1\} * \max(|g|)$, and thus 2 bits are sufficient to encode a gradient. Instead, QSGD uses N bits to map each gradient to a uniformly distributed discrete set containing 2^N bins. Please note TernGrad does not quantize the last classification layer to keep good performance [29], while we sparsify the entire gradients.

Algorithm Setup: Regarding Top-k and FFT based sparsification, results from Figure 13 and [5] show a noticeable convergence slowdown after $\theta > 90\%$. To maintain a reasonable accuracy, we choose $\theta = 85\%$ for both top-k and FFT based sparsification. We use $\min = -1$ and $\max = 1$ as the boundaries, and 10 bits in initializing our N-bit quantizer. Therefore, the compression ratio for Top-k is $1/(1-\theta) = 6.67x$ and FFT based is $21.3x$ with an additional $32/10$ from quantizers. Terngrad uses 2 bits to encode a gradient, while we use 8 bins (3 bits) for QSGD to encode a gradient. As a result, the compression ratio of Terngrad is $16x$ and QSGD is $10.6x$. Please note we calculate the compression ratio w.r.t gradients as gradient exchanges dominate communications in BSP. Following a similar setup in Figure 13, each algorithm is set to run 180 epochs on CIFAR10 and 70 epochs on ImageNet using 8 GPUs.

Figure 14 demonstrates that our framework outperforms QSGD, Terngrad, and Top-k in both the final accuracy and the training wall time on an 8 GPU cluster, and Table 2 summarizes the test accuracy and speedup over the lossless SGD. Particularly, FFT consistently reaches a similar accuracy to SGD with the highest speedup. To further investigate the algorithmic and system advantages of the FFT method, we investigate the gradient quality and the scalability of iteration throughput.

4.2.1 The algorithmic advantages of FFT. We claim the algorithmic advantages of FFT for preserving the original gradient distribution and rendering fewer reconstruction errors than others. We uniformly sampled the gradients of ResNet32 every 10 epochs during the training. Figure 15 demonstrates the distribution of reconstructed gradients w.r.t the gradients before the compression. FFT is the only one that retains the original gradient distribution, though $\theta = 85\%$ frequency has been removed. In contrast, Top-k loses the peak for eliminating the near-zero elements at the same θ . Similarly, QSGD presents 7 clusters for using 8 bins to represent a gradient; and, in general, TernGrad shows 3 major clusters around $\{0, -0.05, 0.05\}$ for using a quantization set of $\{-1, 0, 1\}$. Please

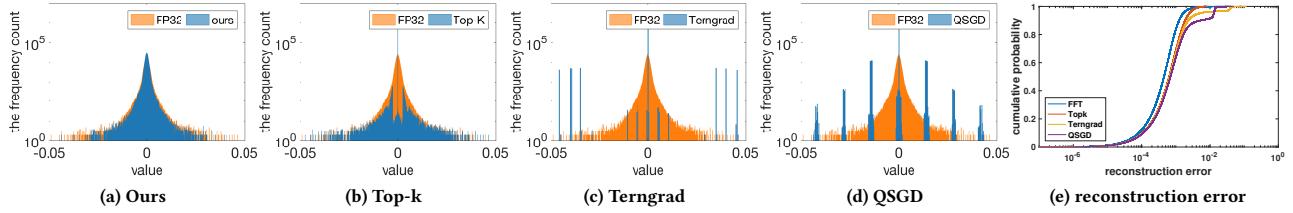


Figure 15: (a)–(d): Histogram of reconstructed gradients (blue) by FFT ($\theta = 0.85$), Top-k ($\theta = 0.85$), QSGD and Terngrad v.s. the original. The reconstructed gradients by FFT is the closest to the original(FP32). (e) Cumulative error distribution of $|g_i - \hat{g}_i|$, where g_i is the i -th true gradient, and \hat{g}_i is the i -th sparsified gradient. FFT incurs less errors than other approaches for 99.7% of the gradients.

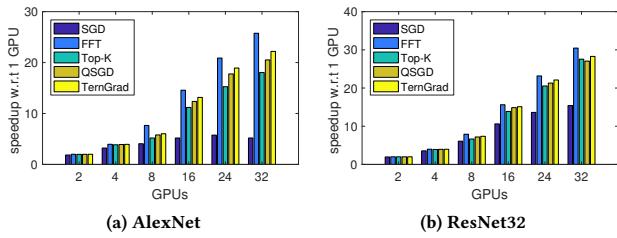


Figure 16: Weak scalability from 2 to 32 GPUs: we measure the iteration throughput, and calculate the speedup w.r.t 1 GPU.

note that Terngrad shows 11 bars; this is due to the aggregation of sparsified gradients from each node. Aside from qualitatively inspecting the gradient distribution, we also quantitatively examined the empirical cumulative distribution of the reconstruction error in Figure 15e. FFT demonstrates the lowest error within the range of $[10^{-5}, 10^{-2}]$. Therefore, FFT can reach better accuracy in the same training iterations.

4.2.2 The system advantages of FFT. Our compression framework fully exploits both the gradient sparsity and the redundancy in 32-bit floating point by further quantizing the FFT sparsified gradient. It enables FFT to deliver a much higher iteration throughput than QSGD, TernGrad, and Top-K. Following the same setting in Figures 14, Figure 16 demonstrates the iteration throughput of training AlexNet and ResNet32 from 2 to 32 GPUs. Please note that using a very large θ (e.g., 0.999) can get an impressive speedup, but it also drastically hurts the final accuracy. Here we still use $\theta = 85\%$. The gradients of AlexNet (ImageNet) is around 250 MB, while the gradients of ResNet32 (CIFAR-10) are only 6MB. Therefore, the scalability of AlexNet is generally better than ResNet32. Better results are also observable if using a slow network, e.g., 100MB Gbps. When $\text{GPUs} \leq 4$, the speedup is similar as communications are intra-node through PCI-E. FFT still consistently demonstrates the highest iteration throughput for a better compression ratio when GPUs increase from 8 to 32.

5 RELATED WORK

We categorize the existing lossy gradient compression into two groups: (1) *quantization* and (2) *sparsification*.

Quantization: 1-bit SGD [24] is among the first to quantize gradients to alleviate the communication cost in the distributed training. Specifically, it quantizes a 32-bit IEEE-754 float into a binary of $[0, 1]$ to achieve a compression ratio of $32\times$. Though their methods are purely heuristic, and their empirical validations demonstrate a slight loss of accuracy, it shows the possibility to train a network with highly lossy gradients. Subsequently, several quantization methods have been proposed. Flexpoint [18] uses block floating-point encoding based on current gradient/weight values. HOGWILD! [9] quantizes both weights and gradients into 8-bit integers by rounding off floats (i.e., low-precision training); but this idea is largely restricted by the availability of low-precision instruction sets. TernGrad [29] quantizes a gradient as $[-1, 0, 1]*|\max(g)|$, while QSGD [4] stochastically quantizes gradients onto a uniformly discretized set. Both approaches distribute the precision uniformly across the representable range—ignoring both the distribution and the range of the gradients. As we show, gradients follow a normal distribution (Figure 4). In our range-based quantizer, we allocate precision for the range and the distribution of the values to better exploit the limited number of bits. Most importantly, QSGD and TernGrad damage the original gradient distribution due to limited representable values after the quantization (Figure 15). As a result, TernGrad and QSGD incur an observable deterioration in the final accuracy (Table 2).

Sparsification: Aji and Heafield [2] present the very first Top-k gradient sparsification showing that the training can be done with a small accuracy loss by setting the 99% smallest gradients to zeros. Based on the Top-k thresholding, Han et al. [14] propose Deep Compression, which uses heuristics like momentum correction and error accumulation to resolve the accuracy loss in the vanilla Top-k. Please note that these heuristics are orthogonal to our methods and can also be applied to improve ours. Jin et al. [17] propose DEEPSZ, which performs error-bounded lossy compression on the pruned weight. It is a modification of the SZ lossy compression framework [11]. Cédric et al. [22] propose a communication sparsification approach called SPARCMIL. Different from ours, the SPARCMIL focuses on the implementation of MPI collective operations of sparse data. D. Alistarh et al. [5] analyze the convergence of Top-k compression. With [5], we noticed a significant convergence slowdown at a large sparsity. As we investigated, these Top-k methods also distort the gradient distribution at a large sparsity, yielding higher approximation errors than the original gradients. At the same sparsity (θ), our FFT method is much better at preserving the

original gradient distribution and shows less approximation error and better results.

6 CONCLUSION

As indicated in Sec. 2, exchanging gradients is the major bottleneck for the distributed DNN training. To alleviate this communication bottleneck, this paper proposes a lossy gradient compression framework that uses an FFT-based gradient sparsification and a range-based, variable-precision, floating-point representation. We theoretically prove that our techniques preserve the convergence and the final accuracy by adapting the sparsification ratio θ during the training, and empirically verify the assumptions and the theory.

At the same sparsification ratio (θ), we show FFT preserves more gradient information than other state-of-the-art lossy methodologies including Top-K sparsification, Terngrad, and QSGD. Besides, our adaptive float quantization further improves the overall compression ratio with negligible loss of gradient information (Fig. 15). These advantages enable us to use a larger compression ratio in retaining the same accuracy as the lossless SGD, than other lossy methodologies to improve the scalability (Fig. 16) in the distributed training.

Our lossy gradient compression framework demands a highly efficient allreduce that supports communications of sparse data, while current MPI implementations, such as Open MPI or MVAPICH, lack the support of sparse collectives. Though this work uses all-gather to circumvent this issue, future research and development of a bandwidth-efficient allreduce with the sparse support are highly desired to facilitate the deployment of lossy gradient compression techniques in practice.

7 ACKNOWLEDGEMENT

The work done by George Bosilca is supported by the National Science Foundation project SI2-SSI: EVOLVE, under Award Number 1664142.

REFERENCES

- [1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*.
- [2] Alham Fikri Aji and Kenneth Heffield. 2017. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021* (2017).
- [3] Tolu Alabi, Jeffrey D Blanchard, Bradley Gordon, and Russel Steinbach. 2012. Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithms (JEA)* 17 (2012), 4–2.
- [4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Advances in Neural Information Processing Systems*.
- [5] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. 2018. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*. 5975–5985.
- [6] Ammar Ahmad Awan, Khaled Hamidouche, Jahanzeb Maqbool Hashmi, and Dhaleswar P Panda. 2017. S-Caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.
- [7] Alex Berson. 1992. *Client-server architecture*. Number IEEE-802. McGraw-Hill.
- [8] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).
- [9] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. 2015. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*. 2674–2682.
- [10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*.
- [11] S. Di and F. Cappello. 2016. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.
- [12] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*.
- [13] Saeed Ghadimi and Guanghui Lan. 2013. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization* 23, 4 (2013), 2341–2368.
- [14] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- [16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*.
- [17] Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. 2019. DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression (HPDC '19). Association for Computing Machinery, New York, NY, USA, 159–170. <https://doi.org/10.1145/3307681.3326608>
- [18] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J Pai, and Naveen Rao. 2017. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 1742–1752.
- [19] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiung Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*.
- [20] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).
- [21] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. 2009. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization* 19, 4 (2009), 1574–1609.
- [22] Cédric Renggli, Dan Alistarh, and Torsten Hoefler. 2018. SparCML: High-Performance Sparse Communication for Machine Learning. *CoRR abs/1802.08021* (2018).
- [23] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. 2016. vdNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*.
- [24] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Fifteenth Annual Conference of the International Speech Communication Association*.
- [25] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*.
- [26] Linnan Wang, Yi Yang, Renqiang Min, and Sriram Chakradhar. 2017. Accelerating deep neural network training with inconsistent stochastic gradient descent. *Neural Networks* 93 (2017), 219–229.
- [27] Linnan Wang, Jimmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. 2018. Superneurons: Dynamic GPU Memory Management for Training Deep Neural Networks. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.
- [28] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*. 1306–1316.
- [29] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*.
- [30] Yiyang Zhao, Linnan Wang, Wei Wu, George Bosilca, Richard Vuduc, Jimmian Ye, Wenqi Tang, and Zenglin Xu. 2017. Efficient Communications in Training Large Scale Neural Networks. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*.

LEMMA .1. Assume $\eta_t \leq \frac{1}{4L}$, $\theta_t^2 \leq \frac{1}{4}$. Then

$$\frac{\eta_t}{4} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \mathbb{E}[f(x^t)] - \mathbb{E}[f(x^{t+1})] + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t}. \quad (11)$$

Proof. By Lipschitz continuity,

$$\begin{aligned} f(x^{t+1}) &= f(x^t - \eta_t \hat{v}_t) \\ &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t \hat{v}_t \rangle + \frac{L}{2} \|\eta_t \hat{v}_t\|^2 \\ &= f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \langle \nabla f(x^t), -\eta_t (\hat{v}_t - v_t) \rangle + \frac{L}{2} \|\eta_t \hat{v}_t\|^2 \\ &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \eta_t \|\nabla f(x^t)\| \|\hat{v}_t - v_t\| + \frac{L}{2} \|\eta_t \hat{v}_t\|^2 \\ &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \eta_t \|\nabla f(x^t)\| \|\hat{v}_t - v_t\| + \frac{L}{2} \eta_t^2 \|v_t\|^2 \\ &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} \|\hat{v}_t - v_t\|^2 + \frac{L}{2} \eta_t^2 \|v_t\|^2 \\ &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t \theta_t^2}{2} \|v_t\|^2 + \frac{L}{2} \eta_t^2 \|v_t\|^2 \\ &= f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} (L\eta_t + \theta_t^2) \|v_t\|^2 \end{aligned}$$

Note that conditioning on x^t , $\mathbb{E}[v_t | x^t] = \nabla f(x^t)$. Then take expectation on both sides, and use the relationship that

$$\begin{aligned} \mathbb{E}[\|v_t\|^2 | x^t] &= \mathbb{E}[\|v_t - \nabla f(x^t)\|^2 | x^t] + \|\mathbb{E}[v_t | x^t]\|^2 \\ &= \mathbb{E}[\|v_t - \nabla f(x^t)\|^2 | x^t] + \|\nabla f(x^t)\|^2 \\ &\leq \frac{\sigma^2}{b_t} + \|\nabla f(x^t)\|^2 \end{aligned}$$

We have

$$\begin{aligned} \mathbb{E}[f(x^{t+1}) | x^t] &\leq f(x^t) + \langle \nabla f(x^t), -\eta_t \mathbb{E}[v_t | x^t] \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} (L\eta_t + \theta_t^2) \mathbb{E}[\|v_t\|^2 | x^t] \\ &\leq f(x^t) - \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} (L\eta_t + \theta_t^2) \left(\frac{\sigma^2}{b_t} + \|\nabla f(x^t)\|^2 \right) \\ &= f(x^t) - \frac{\eta_t}{2} (1 - L\eta_t - \theta_t^2) \|\nabla f(x^t)\|^2 + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t} \end{aligned}$$

Note that $L\eta_t \leq 1/4$, $\theta_t^2 \leq 1/4$, and take expectation over the history, we get

$$\mathbb{E}[f(x^{t+1})] \leq \mathbb{E}[f(x^t)] - \frac{\eta_t}{4} \mathbb{E}[\|\nabla f(x^t)\|^2] + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t}$$

The lemma is proved. \square