FFT-BASED GRADIENT SPARSIFICATION FOR THE DISTRIBUTED TRAINING OF DEEP NEURAL NETWORKS

ABSTRACT

The performance and efficiency of distributed training of Deep Neural Networks (DNN) highly depend on the performance of gradient averaging among all participating nodes, which is bounded by the communication costs. There are two major solutions to reduce communication overhead: one is to overlap communications with computations (lossless), and the other is to reduce communications (lossy). The lossless solution works well for linear neural architectures, e.g. VGG, AlexNet, but the latest networks such as ResNet and Inception render the limited opportunity for such overlapping. Therefore, researchers have paid more attention to lossy methods. 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 techniques are dynamic, which strikes a balance between compression ratio, accuracy, and computational overhead, and highly optimized to maximize performance in heterogeneous environments. We also sketch proves to show how the FFT sparsification θ, the ratio of dropped information, affects the convergence and accuracy, and show that our method is guaranteed to converge using a diminishing θ in the training. 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.

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). This opens up new opportunities to alleviate the communication overhead, by aggressively compressing gradients. One approach to compress the gradients is *quantization*. For example, Terngrad (Wen et al., 2017) maps a gradient into [-1, 0, 1], and QSGD (Alistarh et al., 2017) stochastically quantizes gradients onto a uniformly discretized set larger than that of Terngrad. Such coarse approximation not only incurs large errors between the true and quantized gradients as we demonstrate in Figure 15 [QSGD, TernGrad], but also ignores exploiting the bit efficiency in the quantization (Figure 7). Another approach to gradient compression, *sparsification*, only keeps the top-k largest gradients (Han et al., 2015; Aji & Heafield, 2017; Alistarh et al., 2018). Similarly, Top-k loses a significant amount of true gradients around zeros to achieve the high compression ratio (Figure 15, [Top-k]). In summary, existing lossy methods greatly drop gradients, incur large approximation errors (Figure 15e), leading to deteriorations in the final accuracy (Table 1). To avoid compromising the convergence speed, both *quantization* and *sparsification* must limit compression ratio, leading to sub-optimal improvement of the end-to-end training walltime.

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 for removing the redundant information, while still preserving most relevant information (Figure 15 [FFT]). As a result, FFT incurs fewer errors in approximating the original gradients (Figure 15e), thereby better in accuracy than QSGD, TernGrad, and Top-K (Table 1). 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.

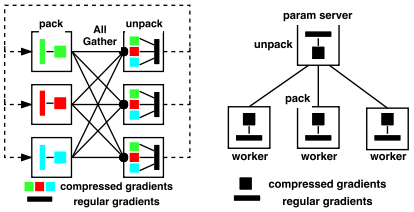
Lastly, our compression framework also delivers high performance. The primitive algorithms in our compression scheme, such as FFT, top-k select, and precision conversions, are efficiently parallelizable and thus GPU-friendly. We resort to existing highly optimized GPU libraries such as cuFFT, Thrust, and bucketSelect (Alabi et al., 2012), 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 to be beneficial in very fast networks, such as current and future 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 precisions floating-point that allocates precision according to the gradient distribution;
- the convergence proof of our methods, and its guidance in selecting compression hyper-parameters, e.g. drop out ratio θ, to ensure the convergence.
- 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 (Dean et al., 2012). It demands extensive intra-DNN communications in addition to gradient exchanges. This largely restricts the training performance, and thereby *Model Parallelism* is often applied in scenarios where the DNN cannot fit onto a computing node (Dean et al., 2012). The second approach, *Data Parallelism* (Wang et al., 2017), partitions the image batch, and



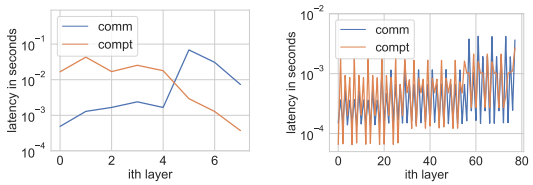
(a) BSP (b) PS

Figure 1. **Two parallelization schemes of distributed DNN training**:(a) Bulk Synchronous Parallel (BSP) strickly 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.

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 (Wang et al., 2018), MXNet (Chen et al., 2015), Caffe (Jia et al., 2014), and TensorFlow (Abadi et al., 2016) 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 (Abadi et al., 2016), MXNet (Chen et al., 2015), and PaddlePaddle implement distributed DNN training with a Parameter Server (PS) (Li et al., 2014). 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 (Berson, 1992) 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 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 tradeoffs between the BSP and PS schemes, with PS having better fault tolerance, and *allreduce* better exploits the network bandwidth, but, as we argue below, in both cases the communication cost is significant, and reduc-



(a) AlexNet (b) ResNet32

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).

ing it can yield substantial gains in training latency.

2.1 Communication Challenges in Distributed Training of DNNs

Communications for averaging sub-gradients of all workers is widely recognized as a major bottleneck in scaling DNN training(Zhao et al., 2017; Dean et al., 2012; Wang et al., 2017). With increasing data complexity and volume, and with emerging non-linear neural architectures, we have identified two critical issues that exacerbate the impact of communications in the scalability and efficiency of distributed DNN training with data parallelism: I) *increasing bandwidth requirements*, and II) *the decreasing opportunity to overlap computation and communication*.

Challenge I: Very large bandwidth requirements during training DNNs are extremely effective at modeling complex nonlinearities 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 ring-based allreduce 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 sizes [1] still consumes $64.17\%, 18.62\%, 33.07\%, 43.96\%$ 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 the gradient averaging of i^{th} layer under 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 (Awan et al., 2017; Rhu et al., 2016), as these networks utilize large convolution kernels to process input data. Figure 2a demonstrates the computation time of the convolution layers is $10\times$ larger than the communication time, easy for overlapping. However, the opportunity for the computation and communication overlapping is very limited in recent neural architectures, such as Inception-V4 (Szegedy et al., 2017) and ResNet (He et al., 2016). The sparse fan-out connections in the Inception Unit (Figure 1a in (Wang et al., 2018)) 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 layerwise computational cost of ResNet is similar to communication (Figure 2b), much harder to overlap than AlexNet.

These two challenges – increasing data exchanged, and decreasing opportunity to hide communication latency – make it attractive to look for solutions that instead decrease the communication volumes. Training a neural network with imprecise gradient updates still works as parameters are iteratively refined (Aji & Heafield, 2017). Particularly, lossy gradient compression can achieve greater compression rates and still allow the network to achieve target accuracies (Alistarh et al., 2017). 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.* (Seide et al., 2014; Wen et al., 2017; Alistarh et al., 2017; De Sa et al., 2015)), where these are represented with lower precision numbers, and sparsification (*e.g.* (Aji & Heafield, 2017; Alistarh et al., 2018; Wangni et al., 2018)), 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. First, we linearize gradients by re-arranging gradient tensors into a 1-d vector for Fast Fourier Transform (FFT). Second, we truncate the gradient frequencies based on their magnitudes to sift out the low-energy frequency components. Third, we 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. Finally, the compressed gradient frequency vector is transferred out. On the receiver side, it follows the same procedure but in the reverse operation and reverse order to decompress the gradient frequency vector into gradients. Detailed discussions of compression components and their motivations are as follows.

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

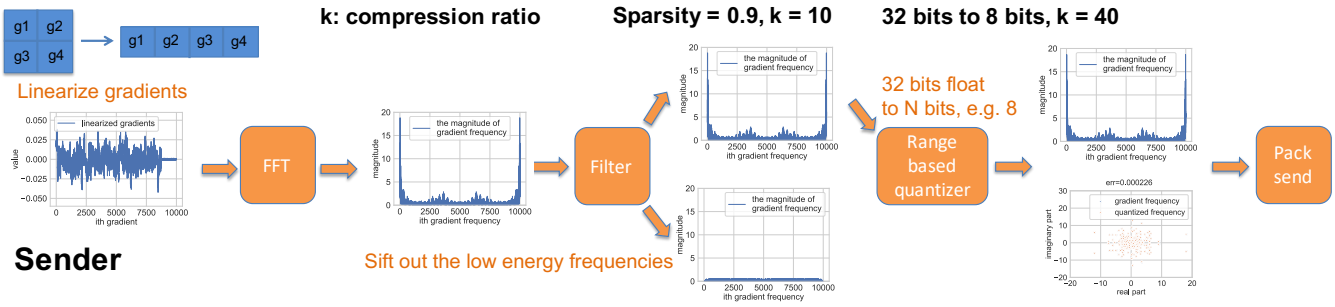
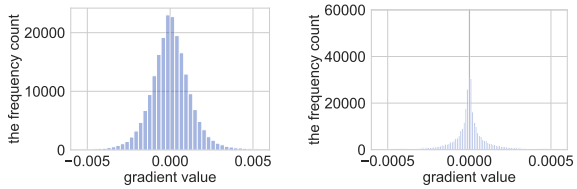
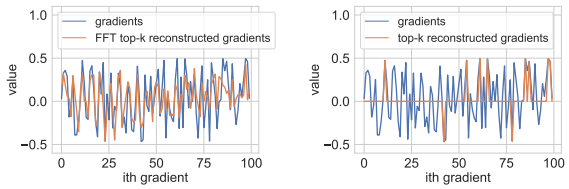


Figure 3. The gradient compression pipeline.



(a) CIFAR-10, ResNet (b) ImageNet, AlexNet

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



(a) FFT Top-k (b) Top-k

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

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 (Han et al., 2015; Aji & Heafield, 2017; Alistarh et al., 2018) have also shown the possibility to train DNNs with only top 10% largest gradients. But 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. $g_i = \sum_{n}^{top_k} x_n e^{\frac{-i2\pi kn}{N}}$, 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 apporach: The detailed computation steps of our FFT sparsification are highlighted by Figure 3. Recent generations of NVIDIA GPUs support mixed precisions; and computing with half-precision increases the FFT throughput up to 2×. So, we convert 32-bit (full-precision) gradients into 16-bit (half-precision) gradients to improve the throughput, and the information loss from the conversion is negligible due to the bounded gradients. 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 Thrust[2] and cuFFT [3] provide highly optimized FFT and sorting kernels for the GPU architecture, we adopted them in our implementations.

Thresholding gradient frequencies yield 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:

- First, we 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 $status = [1, 0, 1, 0, 1, 0, 0]$).
- Second, we perform a parallel prefix-sum on $status$ to generate a $location$ vector ($[1, 1, 2, 2, 3, 3, 3]$).

[2]https://developer.nvidia.com/thrust
[3]https://developer.nvidia.com/cufft

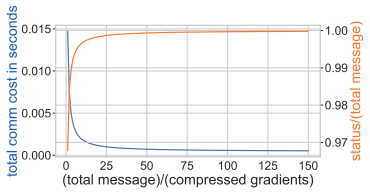


Figure 6. **the effect of status vector**: given 100 MB gradients, the improvement after dropping $> 90\%$ gradients ($\theta = 0.9$) is limited.

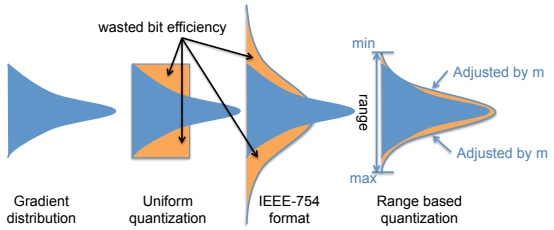


Figure 7. **comparisons of quantization schemes**: the uniform distribution and IEEE 754 format.

- Third, if $status[i] == 1$, we 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 $\theta < 5\%$ is not desired.

3.1.2 Range based Quantization

Motivation: the range of single precision IEEE-754 floating point is $[-3.4 * 10^{38}, +3.4 * 10^{38}]$, 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 true gradient distribution is far from the uniform, thereby it is also inefficient as shown in Figure 7.

Our apporach: 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 ($pbase$) + 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 precisions 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 precisions:

- 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 precisions.
- 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 number. 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 stey-by-step conversion between IEEE 754 and 8 bits our representation.

Figure 9 shows the resulting number distributions of our approach when the range is set to [-0.5, 0.5], and [-5, 5].

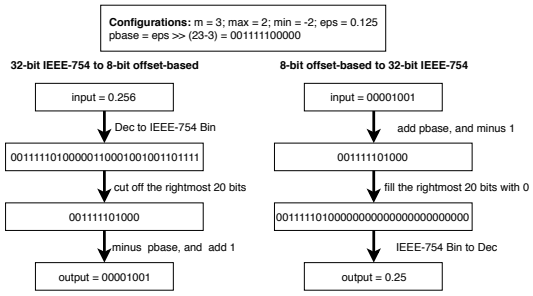
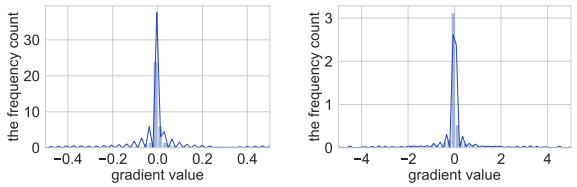


Figure 8. ***Illustration of range based quantizer***: an example conversion of between 32 bits IEEE 754 and 8 bits our representation.

Algorithm 1: Offset-based N-bit floating point

Input: init(min, max)
$pbase\_binary = eps >> (23 - m)$;
Input: 32bit_to_Nbit(32bit_float)
if *32bit_float* $> max$ **then**
  32bit_float = max;
$32bit\_binary = 32bit\_float >> (23 - m)$;
$Nbit\_binary = 32bit\_binary - pbase\_binary + 1$;
Input: Nbit_to_32bit(Nbit_binary)
$32bit\_binary = Nbit\_binary + pbase\_binary - 1$;
$32bit\_float = 32bit\_binary << (23 - m)$;



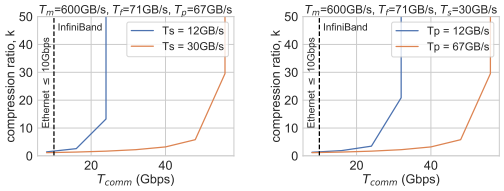
(a) (-0.5, 0.5) (b) (-5, 5)

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

This shows our approach successfully adjusts representation ranges, while still 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, it is easy to achieve the high-performance.

3.2 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 network bandwidth toward the performance. We can define: k as the overall compression ratio; T_m as the throughput of precision change and thresholding—we use the same notation as they are O(N) algorithms and embarrassingly parallel; T_f as the throughput of FFT; T_p as the throughput of packing; T_s as the throughput of Top-k selection; and T_{comm} as the throughput of communications.



(a) Select (b) Pack

Figure 10. **Minimal compression ratio k exhibits performance benefits** at different network bandwidths T_{comm}, packing throughput T_p and selection throughput T_s. k is the overall compression ratio; T_m is the throughput of quantization and thresholding. We assume similar costs on both O(N) operations as they are embarrassingly parallel; T_f is the throughput of FFT. It is easy to get performance improvement from a slow network, while it requires faster compression primitives to be beneficial on a fast network.

Given a message of size M, the cost of compression is $cost_{comp} = M(\frac{4}{T_m} + \frac{1}{T_f} + \frac{1}{T_p} + \frac{1}{T_s})$ (the notations are defined in Figure 10). The compression saves $saved\_cost_{comm} = \frac{M}{T_{comm}}(1 - \frac{1}{k})$. $2cost_{comp} < saved\_cost_{comm}$ must hold to acquire the practical performance gain, $k > \frac{1}{1-2T_{comm}(\frac{4}{T_m}+\frac{1}{T_f}+\frac{1}{T_p}+\frac{1}{T_s})}$. The performance of T_m depends on the GPU DRAM bandwidth, and T_f depends on cuFFT. Therefore, we vary T_p, T_s and T_m in $\frac{1}{1-2T_{comm}(\frac{4}{T_m}+\frac{1}{T_f}+\frac{1}{T_p}+\frac{1}{T_s})}$ to calculate the minimal k. Figure 10 shows that k is extremely sensitive to T_p and T_s as $T_{comm} > 4$, indicating that the performance of compression primitives is critical to a high-end network like InfiniBand.

3.3 Convergence Analysis

Here we present the convergence analysis of the proposed techniques. We formulate the DNN training as:

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

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 $t \to \infty$, 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 $\{1, ..., N\}$, 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 (Nemirovski et al., 2009) (Ghadimi & Lan, 2013). 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 quant(FFT-sparsification(v_i)), so $\hat{\bar{v}} = \sum_1^p \hat{v_i}$. We assume there exists a α such that:*

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

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:

$$x^{t+1} = x^t - \eta_t(\frac{1}{P}\sum_1^p \hat{v_i}) = x^t - \eta_t\hat{\bar{v}}_t. \tag{2}$$

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*

$$\frac{\eta_t}{4}\mathbb{E}[\|\nabla f(x^t)\|^2] \leq \mathbb{E}[f(x^t)] - \mathbb{E}[f(x^{t+1})] + (L\eta_t + \theta_t^2)\frac{\eta_t\sigma^2}{2b_t}. \tag{3}$$

Please check the supplemental material for the proof of this lemma. Summing over (3) for K iterations, we get:

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

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 (4), we get the theorem. □

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 dropout ratio, θ_t, if $\theta_t^2 = L\eta_t$.*

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

$$\mathbb{E}[\|\nabla f(x_{out})\|^2] = \frac{\sum_{t=0}^{K-1}\eta_t\mathbb{E}[\|\nabla f(x^t)\|^2]}{\sum_{t=0}^{K-1}\eta_t} \tag{5}$$

$$\leq \frac{4(f(x^0) - f(x^*)}{\sum_{t=0}^{K-1}\eta_t} + \frac{\sum_{t=0}^{K-1}(L\eta_t + \theta_t^2)2\eta_t\sigma^2}{b\sum_{t=0}^{K-1}\eta_t} \tag{6}$$

Note that $\sum_{t=0}^{K-1}\eta_t \to \infty$, 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 $\mathbb{E}[\|\nabla f(x_{out})\|^2] \to 0$. □

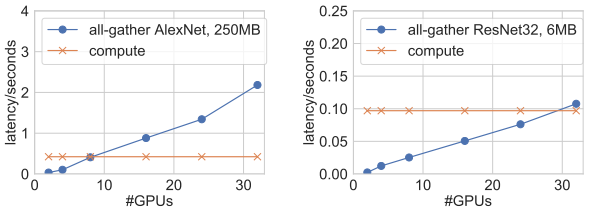
4 EVALUATION

Our experiments consist of 2 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. Then, we show that FFT-based method demonstrates better convergence and faster speed than QSGD (Alistarh et al., 2017), TernGrad (Wen et al., 2017), Top-k sparsification (Lin et al., 2017; Alistarh et al., 2018), 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.

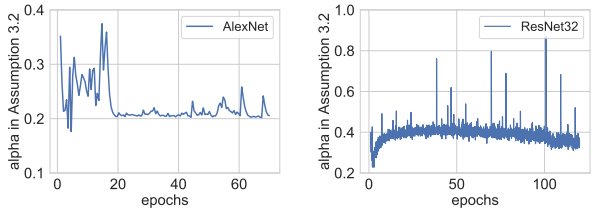
Implementation: we implemented our approach, QSGD, Top-K, and TernGrad in a C++ DL framework, SuperNeurons (Wang et al., 2018); We used allgather in NVIDIA NCCL2 to exchange compressed gradients since existing MPI libraries lack the support for sparse all-reduce (Figure 1a). Every GPU has a copy of global 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.

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.



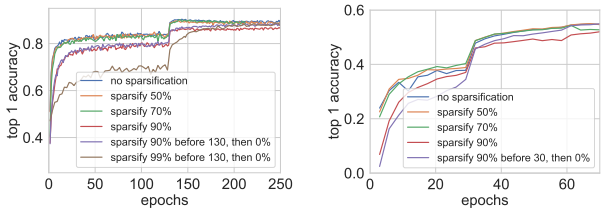
(a) AlexNet (b) ResNet32

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



(a) AlexNet (b) ResNet32

Figure 12. Empirical verification of Assumption 3.2.



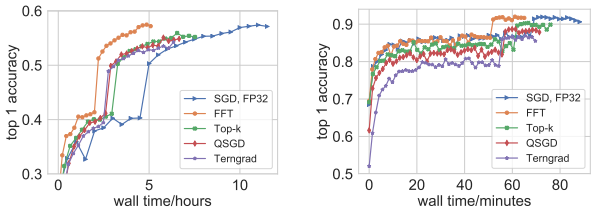
(a) ResNet32 (b) AlexNet

Figure 13. Empirical validation of Theorem 3.5.

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 *all-gather* cost almost linearly increases with the number of GPUs. This is because the total exchanged messages in *all-gather* linearly increase with #GPUs (Gabriel et al., 2004). It demonstrates the necessity of compression. In our experiments, we used 8 GPUs in evaluating the distributed training, and up to 32 GPUs in evaluating the iteration throughput.

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 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.



(a) AlexNet (b) ResNet32

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.

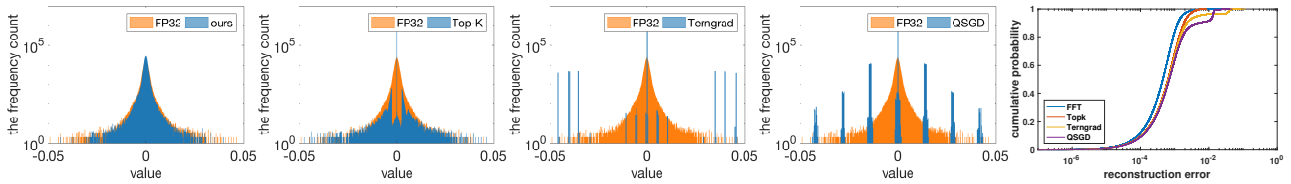
Table 1. **Summarization of Figure 14**: the difference of test accuracy and the speedup over SGD.

Method	AlexNet top1 acc	Speedup w.r.t SGD	ResNet32 top1 acc	Speedup w.r.t SGD
SGD	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

$\bar{v}_1 = [-0.3, 0.5]$ 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 $\alpha = \frac{\|\bar{v}-\bar{\tilde{v}}\|}{\|\bar{v}\|}$. From Figure 13, $\alpha \in [0, 1]$ practically sustaining Assumption 3.2.

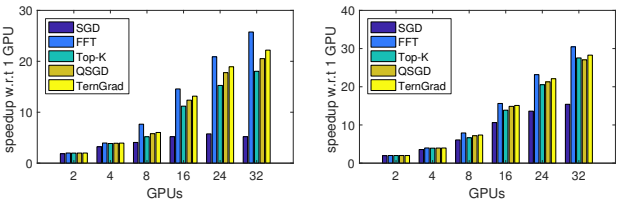
Validation of theorems: Theorem 3.4 states a large compression ratio, i.e. large θ, can jeopardize the convergence. The goal of optimization is to find a local optimum, where the gradient approximates to zero, i.e, $\mathbb{E}[\|\nabla f(x^t)\|^2] \to 0$, as $K \to \infty$. From the inequality in theorem 3.4, $\frac{4(f(x^0)-f(x^{K-1}))}{K} \to 0$ as $K \to \infty$, 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 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 (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.

Theorem 3.5 states that our FFT-based sparsified SGD is guaranteed to converge with a diminishing compression ratio. The theorem compensates for Theorem 3.4, indicating that a large θ can still deliver the same accuracy as SGD if



(a) Ours (b) Top-k (c) Terngrad (d) QSGD (e) reconstruction error

Figure 15. **(a)**→**(d)**: Histogram of reconstructed gradients (green) 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.



(a) AlexNet (b) ResNet32

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

we shrink the θ in the training. In practice, we can shrink θ along with the learning rate η for the condition of $\theta_t^2 = L\eta_t$. Empirical results in Figure 13 validate Theorem 3.5. For example, by setting $\theta = 0.9$ (drop 90%), all of the networks fail to converge to the same case of SGD. However, the compression achieves the same result as 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.

4.2 Algorithm Comparisons

Choice of Algorithms: Here we evaluate our FFT-based techniques against 3 major gradient compression algorithms, Top-k sparsification (Lin et al., 2017; Alistarh et al., 2018; Aji & Heafield, 2017), and Terngrad (Wen et al., 2017) and QSGD (Alistarh et al., 2017). 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-θ), where θ is the drop-out ratio. Please note that Top-k variant e.g. DGC (Lin et al., 2017) 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 maps 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 (Wen et al., 2017), while we sparsify the entire gradients.

Algorithm Setup: Regarding Top-k and FFT based sparsification, results from Figure 13 and (Alistarh et al., 2018) show an obvious convergence slowdown after $\theta > 90\%$. To maintain a reasonable accuracy, we choose θ = 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-θ) = 6.67x and FFT based is 21.3x with 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.

4.2.1 *Result analysis*

Figure 14 demonstrates 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 1 summarizes the performance. Particularly, FFT consistently reaches a similar accuracy to SGD with the highest speedup. To further investigate the algorithmic and system advantages of FFT method, we investigate the gradient quality and the scalability of iteration throughput.

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 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.

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. This 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 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 GPUs ≤ 4, the speedup is similar as communications are intra-node through PCI-E. FFT still consistently demonstrate 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 (Seide et al., 2014) 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×. Though their methods are purely heuristic, and their empirical validations demonstrate a slight loss of accuracies, it shows the possibility to train a network with highly lossy gradients. Subsequently, several quantization methods have been proposed. Flexpoint (Köster et al., 2017) uses block floating-point encoding based on current gradient/weight values. HOGWILD! (De Sa et al., 2015) 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 (Wen et al., 2017) quantizes a gradient as [-1, 0, 1]$*|max(g)|$, while QSGD (Alistarh et al., 2017) 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 precisions 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 1).

Sparsification: Aji and Heafield (Aji & Heafield, 2017) 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, DGC (Han et al., 2015) proposes 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. Cédric et al. (Renggli et al., 2018) proposes a communication sparsification approach called SPARCML. Different from ours, the SPARCML focuses on the implementation of MPI collective operations of sparse data. D. Alistarh et al. (Alistarh et al., 2018) analyze the convergence of Top-k compression. Both (Alistarh et al., 2018) and we noticed a significant convergence slowdown at a large sparsity. As we analyzed, 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

Communication of gradients is a major bottleneck for training DNNs in large-scale computers. To improve the distributed training efficiency, we propose a compression framework using a sparsification method in the frequency domain, and a range-based, variable-precision, floating-point representation to quantize and compress the gradient frequencies after sparsification. We proved theoretically that our compression methodology preserves the convergence and the accuracy of the DNN. It also provides enough data parallelism to exploit the computation power of current GPUs. Experimental evaluation on a distributed heterogeneous platform has shown that the our method effectively improves the scalability of DNN training without compromising the accuracy.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

Aji, A. F. and Heafield, K. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

Alabi, T., Blanchard, J. D., Gordon, B., and Steinbach, R. Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithmics (JEA)*, 17: 4–2, 2012.

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

Alistarh, D., Hoefler, T., Johansson, M., Konstantinov, N., Khirirat, S., and Renggli, C. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pp. 5975–5985, 2018.

Awan, A. A., Hamidouche, K., Hashmi, J. M., and Panda, D. K. 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*, 2017.

Berson, A. *Client-server architecture*. Number IEEE-802. McGraw-Hill, 1992.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

De Sa, C. M., Zhang, C., Olukotun, K., and Ré, C. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pp. 2674–2682, 2015.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, 2012.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, 2004.

Ghadimi, S. and Lan, G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., Khosrowshahi, A., Kloss, C., Pai, R. J., and Rao, N. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1742–1752. Curran Associates, Inc., 2017.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Renggli, C., Alistarh, D., and Hoefler, T. Sparcml: High-performance sparse communication for machine learning. *CoRR*, abs/1802.08021, 2018.

Rhu, M., Gimelshein, N., Clemons, J., Zulfiqar, A., and Keckler, S. W. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, 2016.

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 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, 2014.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

Wang, L., Yang, Y., Min, R., and Chakradhar, S. Accelerating deep neural network training with inconsistent stochastic gradient descent. *Neural Networks*, 93:219–229, 2017.

Wang, L., Ye, J., Zhao, Y., Wu, W., Li, A., Song, S. L., Xu, Z., and Kraska, T. 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*, 2018.

Wangni, J., Wang, J., Liu, J., and Zhang, T. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pp. 1306–1316, 2018.

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

Zhao, Y., Wang, L., Wu, W., Bosilca, G., Vuduc, R., Ye, J., Tang, W., and Xu, Z. Efficient communications in training large scale neural networks. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, 2017.

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

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

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

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

We have

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

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. □