Abstract—The performance and efficiency of distributed training of Deep Neural Networks highly depend on the performance of gradient averaging among all participating nodes, which is bounded by the communication between nodes. There are two major strategies to reduce communication overhead: one is to hide communication by overlapping it with computation, and the other is to reduce message sizes. The first solution works well for linear neural architectures, but latest networks such as ResNet and Inception offer limited opportunity for this overlapping. Therefore, researchers have paid more attention to minimizing communication. In this paper, we present a novel gradient compression framework derived from insights of real gradient distributions, and which strikes a balance between compression ratio, accuracy, and computational overhead. Our framework has two major novel components: sparsification of gradients in the frequency domain, and a range-based floating point representation to quantize and further compress gradients frequencies. Both components are dynamic, with tunable parameters that achieve different compression ratio based on the accuracy requirement and systems’ platforms, and achieve very high throughput on GPUs. We prove that our techniques guarantee the convergence with a diminishing compression ratio. Our experiments show that the proposed compression framework effectively improves the scalability of most popular neural networks on a 32 GPU cluster to the baseline of no compression, without compromising the accuracy and convergence speed.

I. INTRODUCTION

Parameter Server (PS) and allreduce-style communications are two core parallelization strategies for distributed DNN training. At each iteration, each worker produces a gradient, and both parallelization strategies rely on the communication network to average the gradients across all workers. Normally, the gradient size of current DNNs is at the scale of 10^2 MB, and repeatedly transferring such a large volume of messages over millions of iterations is prohibitively expensive, even with the state-of-the-art networks such as Infiniband. 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 for distributed DNN training.

Recently, several heuristics 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 the gradients. One approach to compress the gradients is quantization. Terngrad [1] maps a gradient into [-1, 0, 1], but this static approach is too aggressive, leading to a significant loss in training accuracy. As an enhancement, QSGD [2] stochastically quantizes gradients onto a uniformly discretized set larger than that of Terngrad. However, the gradient distribution is far from uniform. This inconsistency not only incurs large errors, but also restricts the quantization efficiency. Another approach to gradient compression is sparsification. The large fraction of near-zero terms in gradients motivated Aji and Heafield [3] to remove 99% of the smallest gradients (in absolute value). Unfortunately, zeroed-out gradients ignore the critical sign information of dropout gradients, which can also hurt accuracy.

In this paper, we propose a novel gradient compression framework to overcome the aforementioned limitations of existing approaches. Our design is based on a thorough understanding of gradient distributions across different networks and datasets. We leverage two critical observations: 1) the gradients are bounded and cluster around 0, and 2) the range of their distribution rapidly shrinks as training progresses.

The first finding justifies a sparsification strategy, as most gradients cluster near zero. However, we treat the gradient as a 1D signal, and drop near-zero coefficients in the frequency domain, after an FFT. The FFT decomposes a signal as a sum of sin and cos functions, and deleting some frequency components introduces magnitude errors, but the signal maintains trend and sign information. As found in TernGrad, the signs of gradients are critical for accuracy. The sparsification in the frequency domain can achieve the same compression ratio as in the spatial domain, but preserving more relevant information.

We further introduce a new range-based variable bits 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 follows the distribution of the original gradients. The novel range-based design allows us to fully exploit the precision given limited bits, so that the
approximation error can be further reduced. In general, our compression scheme delivers a dynamic solution to compress the gradient at different training stages. In training ResNet32 on CIFAR and AlexNet on ImageNet, our method yields 10x to 40x gradient reductions without slowing down the convergence and compromising the final accuracy.

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 [4], while we propose a simple yet efficient packing algorithm to transform sparse gradients to a dense representation. Minimizing the computational cost of the compression is crucial for it to be beneficial in very fast networks, such as in current and future Infiniband networks. Our experiments on 16 GPUs have shown compression benefits even in a state-of-the-art networking system – 56Gbps Infiniband networks.

Specifically, the contributions of the paper are as follows:

- a study of the gradient distribution on various data-sets and networks that yield insights for gradient compression.
- a novel FFT-based, tunable gradient sparsification that retains trend and sign information. Most importantly, we have proved that our method ensures the convergence with a diminishing compression ratio.
- a novel range-based 8-bit floating point representation that allocates precision according to the gradient distribution. These two together minimize the approximation error in the quantization.
- a combined compression scheme achieving a dynamic compression ratio that maintains the original training accuracy, and achieves high throughput on GPUs and is beneficial even in state of the art Infiniband networks.

II. BACKGROUND AND MOTIVATION

Deep Neural Networks (DNNs) have emerged as powerful and versatile function approximators [5]. A training iteration using back propagation [6] consists of a forward and a backward pass. The forward pass calculates the loss for the current batch of training data while the backward pass computes gradients for updating parameters. Stochastic Gradient Descent [7] updates the network as follows: \[ w^{t+1} = w^t - \eta^t \nabla \psi_w(d^t) \]
where \( w \) is the parameters, \( \eta \) is the learning rate, and \( \nabla \psi_w \) is the gradient w.r.t a sampled image batch d. \( t \) represents the \( t \)-th iteration. Recent large-scale data-sets (e.g. ImageNet [8] or MSCOCO [9]) and emerging non-linear DNNs [7] have exponentially increased the required computations for the training. ResNet50 [11] takes 14 days to finish a 90-epoch ImageNet-1k training on a NVIDIA M40 GPU, and it costs \( 10^{12} \) FLOPs in total. This, combined with the increasing memory requirements of these networks, makes seeking a parallel solution imperative to sustain the development of large scale DNNs.

Generally, there are two strategies to parallelize the DNN training: Model Parallelism and Data Parallelism. Model Parallelism splits a network into several parts, with each being assigned to a computing node [12]. 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 [12]. The second approach, Data Parallelism [7], partitions the image batch, and every computing node holds 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 [10], MXNet [13], Caffe [14], and TensorFlow [15] 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), or with all-to-all group communications (e.g., allreduce). TensorFlow [15], MXNet [13], and PaddlePaddle [5] implement distributed DNN training with a Parameter Server (PS) [16]. In this distributed framework, the parameter server centralizes the parameter updates, while workers dedicate 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 [17] style design easily supports fault tolerance and elastic scalability, the major downside is the network congestion on the server. The framework claims to alleviate this issue with the asynchronous SGD or stale-SGD [18], but it demands additional iterations to compensate for the declined convergence speed.

Alternatively, allreduce-based Bulk Synchronous Parallel SGD is a better fit for deployments with high-speed, dense interconnects, such as modern Infiniband networks. Instead of using a star topology, allreduce pipelines the message exchanges at a fine-grained 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 an appealing scalability where the cost is independent of the number of computing nodes. In addition, BSP SGD converges faster than ASGD, used by the PS approach [7]. There are tradeoffs between the two approaches, with PS having better fault tolerance, and allreduce requiring dense interconnects, but, as we argue below, in both cases the communication cost is significant, and reducing it can yield substantial gains in training latency.

A. Communication Challenges in Distributed Training of DNNs

Communications for averaging sub-gradients of all workers is widely recognized as a major bottleneck in scaling the DNN

\[1\] We will use images as the training data in this paper, without loss of generality.

\[2\] we classify sequentially connected networks as linear networks, otherwise non-linear. Our definition is consistent with the one in [10].

\[3\] http://paddlepaddle.org
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) very large and increasing bandwidth requirements, and II) increasingly limited 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, as gradients modify parameters to achieve the intended behavior. Specifically, the gradient sizes of AlexNet, VGG16, ResNet50, and InceptionV4 are 250MB, 553MB, 102MB, and 170MB.

In a parameter server scheme, the total data exchanged reaches up to Petabytes in the training. It is a common practice to train on GPU nowadays. The GPU memory usage linearly grows with the batch size [10], [20], [21], so each GPU computes with a small batch due to the constraint of GPU DRAM. (e.g. the single GPU batch of AlexNet is 256, InceptionV4 is 128, VGG16 is 256, and ResNet50 is 32.) A small batch requires more iterations than a large batch to traverse the dataset, thereby more gradient averaging and more communications. To simplify the analysis, let’s assume a GPU utilizes a batch of size $10^3$. Then, $10^3$ GPUs aggregate a batch of size $10^4$. To be fully trained towards ImageNet, AlexNet, and VGG16 need 80 epochs, ResNet50 130 epochs, and InceptionV4, 160 epochs. Note that the convergence of a large batch is slower than that of a small batch [22]. That implies it takes at least 9600 iterations for AlexNet and VGG16, 15600 iterations for ResNet50, and 19200 iterations for InceptionV4 to be fully trained at the batch of $10^4$. The total iterations is the total training images divided by the aggregated batch size. Since an iteration exchanges $\text{size}(\text{gradient}) \times \text{count}(\text{GPUs})$ information, AlexNet, VGG16, ResNet50, InceptionV4 transfer, respectively, at least 0.24, 0.51, 0.16, and 0.33 PetaBytes when training with $10^3$ GPUs. Such intensive communications pose a great challenge even for throughput oriented InfiniBand networks. It takes 56Gbps FDR 23.7 hours to transmit 0.5 PetaBytes with a practical speed of 6GB/s. A state-of-the-art GPU (e.g. NVIDIA Tesla V100) has 900 GB/s of memory bandwidth with HBM2 enabled. This drastic performance gap between GPU and FDR ($10^2$ slower) further makes communication a significant overhead.

Even with the highly optimized ring-based all-reduce, the communication overhead is still nontrivial. Fig. 1 demonstrates the percentages of communications to computations at a moderate batch size. The communication of AlexNet, VGG, InceptionV4 and ResNet50 consumes 64.17%, 18.62%, 33.07%, 43.96% of the total time, respectively.

**Challenge II: Increasingly limited 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 technique has proven to be effective on linear networks such as AlexNet and VGG16 [23], [24], and it does not slow down the SGD convergence. We classify a network as linear if the data is sequentially propagated through the network, and non-linear otherwise. For detailed definitions of linear and non-linear neural architectures, please refer to [10]. Linear networks utilize large convolution kernels to process input data, making the computation time of the convolution layers 10x larger than the communication time(Fig 16a and Fig 16d).

However, the opportunity for the computation and communication overlapping is very limited in the more recent non-linear neural architectures, such as ResNet50 [11] and InceptionV4 [25]. The sparse fan-out connections in the Inception Unit (Fig 1a in [10]) replaces one large convolution (e.g. $11 \times 11$ convolution kernel in AlexNet) with several small convolutions (e.g. $3 \times 3$ convolution kernels). ResNet also utilizes either $1 \times 1$ or $3 \times 3$ small convolution kernels. As a result, the computational cost per layer of non-linear neural architectures ($10^{-2}s$) is 10x smaller than linear ones ($10^{-1}s$). Fig 2c and Fig 2d demonstrate that the allreduce time of ResNet50 and InceptionV4 are in the same order of magnitude as the computation for most layers. The limited layer-wise
computations on the emerging non-linear DNN provide limited opportunity for the communication/computation overlapping. In this paper, we focus on compressing down the gradient message size to reduce the cost of communications, while retaining the key gradient information for the convergence guarantee.

B. Compressing Gradients

These two challenges – increasing data exchanged, and decreasing opportunity to hide communication latency – make it attractive to look for solutions that decrease the communication volumes. Congestion at the parameter server caused by large gradient messages prolongs the staleness of gradients [26], hurting the convergence rate. Although the ring based `allreduce` fully utilizes nodes’ bandwidth, large gradients incur non-negligible overhead especially compared to the cost of computations (Fig.1). Training a neural network is an iterative process, and imprecise gradient updates do not significantly affect the final accuracies as parameters are still iteratively refined toward the target behavior [2]. Lossy gradient compression can achieve greater compression rates and still allow the network to achieve target accuracies [2]. 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. [1], [2], [27], [28]), where these are represented with lower precision numbers, and sparsification (e.g. [3]), where small gradient values are treated as zero and not transmitted. We discuss these approaches in detail in Section IV. 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 even in Infiniband networks.

III. METHODOLOGY

A. Intrinsic Gradient Distribution

Gradients $\nabla \psi_w (d')$ change w.r.t network architectures, parameters $w$, and the image batch $d$ at the current training iteration $t$. $\| \nabla \psi_w (d') \|_2$ is bounded if loss is finite. Gradient Clipping [6] is an effective method to stabilize the gradient; it scales the bounded gradient into a range of $[-c_1, +c_1]$, where $c_1$ is a constant (e.g. Caffe uses 35, while MXNet uses 10). Fig.5

Figure 3: Gradient distributions on ImageNet and CIFAR10. We sampled gradients in every $10^3$ and $10^4$ iterations from a full training circle. It is interesting to see gradients follow a normal distribution ($\mu = 0$).

Figure 4: Gradient distributions at different training stages. We evenly sampled 7 millions gradients in iterations [0, 1000] and [7000, 8000]. The experiment demonstrates gradients shrink toward 0 as the training progresses.

B. The Compression Framework

Fig.5 demonstrates the procedures of our gradient compression framework. First, we truncate the gradient frequencies based on their magnitudes to sift out the low energy frequency components. Second, we transform the frequencies based on their magnitudes to sift out the low energy frequency components. The range of single precision IEEE 754 formatted floating point is $[-3.4 \times 10^{38}, +3.4 \times 10^{38}]$, while the range of gradients and its frequency are much smaller (e.g. [-1, +1]). This enables us to represent the bounded gradients with much fewer bits than the single precision float.

Figure 5: The gradient compression scheme.
between the extra computing cost and reduced communication costs. The primitive algorithms we pick exhibit the massive data parallelism, which makes them GPU friendly. The following covers a thorough discussion of each compression components.

1) Removing Redundant Information with FFT and Its Related Issues: We propose doing sparsification in the frequency domain because it preserves the shape of original signal even after removing many frequency components (Fig 7). Fig 6 demonstrates this energy based FFT approach still precisely captures original gradients even removing 70% frequencies. We highlight the FFT sparsification by the leftover box in Fig 5. Recent generations of NVIDIA GPU support mixed precisions, and it increases the FFT throughput up to 2x by computing with the half precision. So, we convert 32 bits gradients into 16 bits half to improve the throughput, and the information loss is negligible due to the bounded gradients. We introduce a new hyper-parameter, $\theta$, to regulate the dropout ratio of frequencies. If $\theta = 0.8$, we keep the top 20% frequency components in the magnitude, and reset the rest to zeros. The selection is implemented with either sorting or top-k. Since Thrust and cuFFT provide highly optimized FFT and sorting kernels toward the GPU architecture, we adopt 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, 3, 3, 3, 3]).
- Third, if status[i] == 1, we write sparse[i] to dense[location[i]] (assume dense vector is the packed result).

This parallel algorithm delivers 689x speedup than the single thread one on TESLA V100 with a throughput of 34 GB/s. We need to send the status vector in addition to the compressed gradient to perform the decompression. Since the status vector is a bitmap that tracks the location of non-zero elements, it’s size toward the performance is very limited.

2) Range based Quantization: In this section, we introduce a method to further compress the packed gradient frequencies vector from 32-bit float to an M-bits floating point number. In designing a compact float point, the most straightforward solution is to follow the IEEE format: reducing the number of bits to represent exponent and mantissa, i.e. the 8 bits 754 formatted floating point use 4 and 3 bits for exponent and mantissa respectively. However, as seen in Fig 5, the range of gradient $[\min, \max]$ is different from the range of the representable number of IEEE format. It is known that the range of the representable number can be adjusted by arranging the number of bits used by exponent and mantissa respectively. However, N-bit IEEE format has only $N - 2$ combinations of exponent-mantissa, regardless which combinations to choose, the range is either too large or too small, both of which compromises precisions. Thus, the traditional IEEE format cannot fully exploit bits in maximizing the precisions.

To dynamically change the range of N-bit floating number, another conventional way is to equally divide the $\max - \min$ into $2^N$. However, as shown in Fig 3, the distribution of gradient is similar to a normal distribution, which requires more representable numbers around 0 and less numbers around $\min$ or $\max$. Therefore, the equal division method still cannot efficiently represent the gradient number.

Algorithm 1: Offset-based N-bit floating point

```
Input: init(min, max)
1. phase = eps >> (23 - min);
2. Input: 32bit_to_Nbit(32bit_num)
3. if 32bit_num > max then
4. 32bit_num >>(23 - m);
5. Nbit_num = 32bit_num - phase + 1;
Input: Nbit_to_32bit(Nbit_num)
6. 32bit_num = Nbit_num + phase - 1;
7. 32bit_num << (23 - m);
```

To solve the number distribution issue, we introduce a novel offset-based representation of N-bit floating point number. The main idea of offset-based representation is to use the N-bit binary format of a positive number as base number $phase$, and encode it to “0...01”(the quantity of 0 is N-1). The all other
numbers are encoded as “0...01 + offset between $p_{\text{base}}$”. The same idea also applies to negative numbers, and “0...00” is reserved for number 0. Therefore, the total $2^N$ representable numbers are: the first number is 0, the next $P$ numbers are positive numbers and the rest $2^N - P$ numbers are negative numbers.

To simplify the tuning algorithm, our approach is to fix the difference between two numbers are increased by 2, the $\min_{\text{eps}}$ our 8-bit ($N=8$) floating number given $P$ this path, than $\min$ actual number of negative number is larger than positive number, then we increase $\min_{\text{eps}}$ so when $\min_{\text{eps}}$ is converging to $max$ or $\min$. It is noticed that such distribution is close to the distribution of gradient number showing in Fig.3 Therefore, our approach is able to dynamically change the range of representable numbers of N-bit floating point number accordingly, and maintain a reasonable precision loss. Besides, our compress and decompress operations are embarrassingly data parallel, therefore, it is able to take the benefit of GPU to boost the performance of data compression.

C. Convergence Analysis

Existing gradient sparsification techniques ([29] and [3]) are empirically verified missing the theoretical analysis. Here we present the convergence analysis of our compression techniques.

The DNN training is equivalent of the following problem:

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

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 $\|\nabla f(x(t))\| \leq \epsilon$ as $t \to \infty$, where $\epsilon$ 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 $\nabla f$ 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: Given a vector $v$ (e.g. gradients), a fraction $0 \leq \theta \leq 1$ to control the percentage of dropping, and a sparsification function $T(v, \theta)$ (e.g. the gradient thresholding), let’s denote $\hat{v} = T(v, \theta)$. Then, we require

$$\|v - \hat{v}\| \leq \theta\|v\|.$$  

and

$$\|\hat{v}\| \leq \|v\|.$$  

Please note $\|\hat{v}\| \leq \|v\|$ is not necessary, but our sparsification techniques, either in the time or frequency domain, satisfy this condition. In addition, the precision error induced by the range based N-bits float is limited; so the assumption is legitimate. For arbitrary sparsification satisfying only $\|v - \hat{v}\| \leq \theta\|v\|$, then instead, we have $\|\hat{v}\| \leq (1 + \theta)\|v\| \leq 2\|v\|$.  

Assumption 3.2: Suppose $j$ is a uniform random sample from $\{1, ..., N\}$, then we make the following bounded variance assumption:

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

for any $x$. This is a standard assumption widely adopted in the SGD convergence proof [30] [31].
For the mini-batch training, the stochastic gradient $v_t$ is

$$v_t = \frac{1}{b_t} \sum_{j \in B_t} \nabla f_j(x^t)$$ (2)

where $B_t$ is the index set of the mini-batch at the $t$-th iteration, having $b_t$ samples. With the gradient sparsification, one SGD update follows:

$$x^{t+1} = x^t - \eta_t \hat{v}_t$$ (3)

with $\hat{v}_t = T(v_t, \theta_t)$ (the compressed gradient), $v_t$ (the original gradient) defined in $T$, and $\eta_t$ as the learning rate. Then, we have the following lemma for one step:

**Lemma 3.3:** Assume $\eta_t \leq \frac{1}{4T}, \theta_t^2 \leq \frac{1}{4}$. Then

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

Please check the supplemental material for the proof of this lemma. Let’s sum over $K$ for $K$ iterations, we get

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

Now, we’re ready to present the convergence theorem.

**Theorem 3.4:** If we choose a fixed learning rate $\eta_t = \eta$ and a fixed dropout ratio $\theta_t = \theta$ in the sparsification function, 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. By the $K$-th iteration, we get the theorem.

The implication of Theorem 3.4 compared with the regular SGD, the gradient compression introduces $(L\eta_t + \theta_t^2)\frac{2\eta_t \sigma^2}{b_t}$ into the bound. This indicates we need to decrease the $\theta$ (dropout ratio) and $\eta$ (learning rate), or increase $b$ (batch size) to tighten the bound to 0; otherwise the training oscillates in the neighborhood of a stationary point, negatively affecting the final accuracy.

**Theorem 3.5:** If we apply the diminishing stepsize $\eta_t$ satisfying $\sum_{t=0}^{\infty} \eta_t = \infty, \sum_{t=0}^{\infty} \eta_t^2 < \infty$. Our compression algorithm guarantees to converge with a diminishing drop out ratio $\theta_t$, 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

$$\mathbb{E}[\|\nabla f(x_{out})\|^2] \leq \frac{\sum_{t=0}^{K-1} \eta_t \mathbb{E}[\|\nabla f(x^t)\|^2]}{\sum_{t=0}^{K-1} \eta_t} \leq \frac{4(f(x^0) - f(x^*))}{\sum_{t=0}^{K-1} \eta_t} + \frac{\sum_{t=0}^{K-1} (L\eta_t + \theta_t^2)2\eta_t \sigma^2}{b \sum_{t=0}^{K-1} \eta_t},$$

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

IV. EVALUATION

In this section, we present the results of our experimental studies that assess the overall gradient compression techniques. Our evaluations are consistent with TernGrad [11] and Deep Gradient Compression (DGC) [29], that exam the convergence (Sec. IV-A1) and iteration throughput (Sec. IV-A3), respectively. We prototype the proposed gradient compression in SuperNeu-
(d) AlexNet, ImageNet, Loss

(f) VGG16, CIFAR

(g) ResNet32, CIFAR

Figure 11: Iteration-wise convergence rate w.r.t FFT drop out ratio \( \theta \). We test every 500 iterations. For resnet32, mixed comp utilizes \( \theta = 0.9 \) before 100th tests, and \( \theta = 0 \) afterwards. For AlexNet, the threshold is the 12th test. \( k \) is the compression ratio.

rons \[^{6}\] parallelized with MPI and PS frameworks. Fig.10 depicts their integrations with the gradient compression. We conduct experiments on Comet cluster hosted at San Diego Supercomputer Center. Comet has 36 GPU nodes, and each node is equipped with 4 NVIDIA TESLA P100 GPU. Nodes are connected via 56Gbps FDR InfiniBand.

A. Evaluation of Overall Compression Framework

The overall compression ratio is contingent upon the frequency drop-out ratio \( \theta \) and the number of bits for quantizations. The compression ratio of FFT dynamically changes w.r.t \( \theta \), and the quantization delivers an additional 4x compression from 32 bits to 8 bits. Therefore, the overall compression ratio \( k \) is

\[
\frac{1}{1 - \text{freqdrop%}}.
\]

The hyper-parameter for quantization is the representation range. The bounded, shrinking gradient allows us to set the range based on sampled gradient frequencies from the first few iterations. In our experiments, the most notable gradient frequencies fall between \([-1, 1]\) on AlexNet, and \([-6, 6]\) on ResNet. We use these ranges as the \textit{min} and \textit{max} boundaries to initialize \( N \) bits floating point quantizer. Any numbers fall beyond the range will be represented by the closest boundary, e.g. -2 is quantized as -1 if the range is \([-1, 1]\).

1) Convergence v.s. Compression Ratios: The only hyper-parameter that is sensitive for tuning is the frequency drop-out ratio \( \theta \) in FFT, which also determines the overall compression ratio. \( \theta \) regulates how much information to discard. Our theoretical analysis in Sec. III-C indicates \( \theta \) is an important factor toward the final accuracy (Theorem 3.4), while a diminishing \( \theta \) guarantees the original training performance (Theorem 3.5). The performance deterioration is observable from the drastic accuracy drop from changing \( \theta \) to 0.9 in Fig.11a and Fig.11g. This is consistent with Theorem 3.4 and our intuition that aggressively filtering out gradients certainly drop useful information, thereby damaging the final accuracy.

Our model agnostic compression techniques guarantee to converge. There are two solutions to fix the accuracy drop: 1) using a small \( \theta \), the accuracy is similar to the case without compression when \( \theta < 40 \) in all models of Fig.11 2) dynamically shrinking \( \theta \) in the training, which is suggested by Theorem 3.5. We start the training with \( \theta = 0.99 \), and change \( \theta \) to 0 after 100th test (ResNet32) and 12th test (AlexNet). The shrunk compression ratio successfully fixes the degraded accuracy of using static \( \theta = 0.99 \) in both Fig.11a and Fig.11g. These results validate the Theorem 3.5. Theorem 3.5 also suggests tuning \( \theta \) is similar to tuning the learning rate for SGD. In practice, not only stepwise \( \theta \) scheduling can be used, and other scheduling policy such as the polynomial and sigmoid decays that are widely implemented for the learning rate scheduling can also be applied, based on datasets and neural networks.

Our compression framework retains the original SGD iteration-wise convergence rate at an appropriate compression ratio. Fig.11 demonstrates the progress of validation accuracies in training AlexNet and ResNet50 on ImageNet, ResNet32 and VGG16 on CIFAR10. In all cases, \( \theta = 0.7 \) achieves the same

\[^{6}\]https://github.com/linnanwang/superneurons-release
validation accuracy as the original without compression. This demonstrates a reduction of 13.4x in gradients, while without slowing down the convergence rate. In an extreme case, we still preserve the original convergence rate at a mixed compression ratio of 400 (Fig.11g and Fig.11h). These experiments suggest that our compression framework is model agnostic, and it maintains the original convergence speed with a proper \( \theta \).

2) Comparisons with Other Gradient Compression Techniques: we have classified existing gradient compression techniques into two categories, quantization and time domain sparsification. Table I demonstrates the main characteristics of each compression algorithms and our distinctions to them.

   a) Quantization: Terngrad \[1\] maps each gradients to the set of \{-1, 0, 1\}, while QSGD \[2\] stochastically quantizes the gradient into a uniform distributed discrete set. Both of them use different quantization methods to reduce the gradient size. Since Assumption 3.1 also holds on these techniques, Theorem 3.2 works for them. Because these methods utilize a static \( \theta \) (not using a diminishing \( \theta \)) in the training, the final accuracy will be negatively affected as demonstrated by the accuracy drop between terngrad and origin Fig.12. b) Time domain Sparsification: Deep Gradient Compression (DGC) \[29\] and \[3\] are two time domain sparsification techniques. In this paper, we propose doing the sparsification in the frequency domain as it preserves the shape of the original signal. We use the same \( \theta \) for sparsifying in either the time or frequency domain, and Fig.12 demonstrates the performance of sparsifying in the frequency domain is better than the time one.

3) Compression Reduces the Iteration Wall Time: a) MPI Parallelization: Fig.13 (a, b) demonstrates the cost break down of a BSP-SGD iteration with and without gradient compression on 16 P100 GPUs. Specifically, AlexNet and VGG16 are able to reduce the communication cost to 1/10 after setting compression ratio to 13. The communication reduction of ResNet50 is about 1/4 since the gradient size of it is small (102MB), while the computation is expensive. Thus, they are not able to reach 1/10 reduction. The cost of compression is also negligible to the saved communication time due to our performance optimizations and the massive parallelism of selected compression algorithms. b) PS Parallelization: In our implementations, one GPU works as a PS receiving gradients from any worker GPUs using MPI_Recv from any sources (pull in PS). Then, PS updates the gradient and send the latest parameters back to the worker GPUs (pull in PS) with MPI_Send. Fig.14 demonstrates that the reduced gradient size successfully alleviates the congestion problem in PS. With the gradient compression, the throughput, i.e. the number of gradient updates on PS in a unit time, has been improved by 3.13x on AlexNet and 1.86x on ResNet50. The improvement is less obvious on ResNet50 (102 MB gradients) than AlexNet (250 MB) due to the small gradient size and their nature of leaning towards computation intensive.

Table I: Overview of existing gradient compression algorithms.

| Methods        | Ideas               | Representation            | Compression Types | Parallel Optimization | Compression Ratio |
|----------------|---------------------|---------------------------|-------------------|------------------------|-------------------|
| Ours           | gradient frequency sparsification | N-bits range based float | Dynamic | Yes | Tunable |
| TernGrad       | gradient quantization | [-1, 0, 1]               | Static            | No | 16x     |
| DGC            | gradient sparsification | 32 bits IEEE-754 float  | Dynamic | No | Tunable |
| QSGD           | stochastic quantization | 4-bits uniformly quantized range | Static | No | 8x     |

Figure 13: Iteration wall time improvement of MPI (a, b) and PS (c, d) parallelization from using the gradient compression (comp) against the one without compression (orig) on 16 P100. comp is the compression cost, comm is communications and compr is computations. \( \theta = 0.7 \).

Figure 14: Scalability of MPI parallelization with/without the gradient compression from 2 \( \rightarrow \) 32 P100 GPUs. DGC stands for Deep Gradient Compression \[29\].

Figure 15: The step-by-step contribution of 3 compression techniques to the iteration time of AlexNet, which consists of computations and communications (MPI).
4) Improvement to the Distributed Training: Fig 14 demonstrates the scalability improvement of having the gradient compression. The compression ratio of Terngrad [1] is 16 as it only needs 2 bits to represent a gradient, while the compression ratio of DGC [29] is 1000x by dropping 99.9% gradients. From the figure, all three techniques have significantly improved the scalability. The speedup of AlexNet on 32 GPUs is 4.65x without the compression, while Terngrad, DGC and ours demonstrate a speedup of 20.18x, 27.50x and 25.12x, respectively. Also, the speedup of ResNet50 on 32 GPUs is 11.3479391 without the compression, and Terngrad, DGC and ours demonstrate a speedup of 27.17x, 31.53x and 29.54x, respectively. These data demonstrate the gradient compression is an effective approach to speed up the distributed DNN training. Although DGC has an impressive compression ratio of 1000x, the scalability improvement to ours is very limited.

B. Components Evaluations

The FFT sparsification, Pack, and Quantizations are three key components in our compression framework. Fig 15 demonstrates the contribution of each component to the overall performance. FFT incurs additional computations as it sets 70% (when $\theta = 0.7$) of low-energy frequencies to zeros. Since FFT only sets frequencies to 0 without removing them from gradients, it does not reduce the size of messages. Instead, Pack significantly alleviates communications as it eliminates zeros from sparsified gradients, thus reduces 70% message size. Furthermore, Quantization provides an additional 4 times message reduction by representing a float from 32 bits to 8 bits. Through these 3 steps, the benefit from the gradient compression significantly outweighs its additional cost, bringing 3x iteration-wise speedup.

C. Improving the communications/computations overlapping

The recent non-linear neural architectures demonstrate limited computations to overlap with communications at each layer (Fig 2). Our techniques also enhance the communications/computation overlapping as shown in Fig 16. This further improves the scalability in training emerging nonlinear neural architectures with this convergence-invasive technique.

V. RELATED WORK

1) Float Compression: Lossy floating point data compression is a technique widely used in scientific computing to alleviate IO pressure. There are two major compression approaches: prediction-based and transformation-based. Prediction-based compression methods such as sz [32] and fzip [33] usually provide better compression rate, but are inherently sequential, and, thus, not amenable to high throughput GPU implementation. Transformation-based compression methods such as ZFP [34] usually provide smaller compression rates. ZFP involves block transform and bit-plane encoding, which can be parallel with CUDA threads and is thus GPU friendly. Even though our compression method delivers lower compression ratio than above approaches, our method is GPU friendly to reduce the overall communication time even on the high-end network and GPUs.

2) Other Gradient Compression Techniques: Several gradient compression approaches have been proposed to alleviate the costly gradient exchange in these distributed training frameworks. We categorize existing works into two groups: quantization and sparsification.

For quantization, 1-bit SGD [27] quantizes a gradient as 0 or 1, while TernGrad [1] utilizes an additional -1, i.e. [-1, 0, 1]. These quantization techniques aggressively drop the gradient precision, failing to capture the critical minor differences among gradients especially at the fine-tuning stage. Thus, these methods suffer from accuracy loss. In addition, 2-bit quantization offers a static compression ratio of up to 16x, while our compression framework allows users to dynamical adjust compression ratio. QSGD [2] stochastically quantizes gradients onto a uniformly discretized set. BuckWild! [28] quantizes weights and gradients into 8 bits integers. Both of these approaches distribute the precision uniformly across the representable range, ignoring both the distribution and the range of the values. As we show, gradients tend to follow a normal distribution (Fig 3 and Fig 4), with most gradients around 0. Our range-based N-bit quantization allocates precision according to both the range and the distribution of the values, better utilizing the limited number of bits.

For sparsification, Aji and Heafield [3] discuss a heuristic gradient sparsification that sets 99% smallest gradients (in absolute value) to zeros. Although this achieves 100x message reduction, setting gradients to zeros ignores other useful information, notably the sign. We propose compressing gradients in the frequency domain by using FFT as a function approximator to the original gradient. This allows us to achieve the same compression ratio as the sparsification in the spatial domain, while still preserving the sign information of dropout gradients to accelerate the convergence (Fig 7).

3) Parameter Compression: this line of research seems to be similar to the gradient compression, but the key difference is that the parameter compression is a static problem. Gradients are constantly changing across iterations, while learned parameters are constant after the training. This static nature enables us to apply expensive compression algorithms, such as
Huffman Coding [35] or Sparse Pruning [36], [37], ignoring the compression overhead, while compressing gradients needs to leverage not only the compression ratio but also the performance. Otherwise, the compression overhead can easily offset the benefits.

VI. CONCLUSION

Communications of gradients are a major bottleneck for training DNNs in large-scale computers, and leads to two critical issues that restrain the efficiency of training DNNs: one is limited network bandwidth for communicating large gradients, the other one is insufficient overlapping of communication and computation in the emerging non-linear neural architectures. To improve the distributed training efficiency, we propose a compression framework by analyzing the data distribution of DNN gradients. Based on the data distribution, we propose a sparsification method in the frequency domain to achieve the same compression ratio as in the time domain, but retaining the trend and sign information. Furthermore, we propose a new range based floating point representation to further compress gradients. Not only is our compression framework designed according to the distribution of DNN gradients, but it also provides enough data parallelism to exploit the computation power of current GPUs. Experiments show that the proposed compression framework improves the scalability of most popular neural networks on a 32 GPU cluster, without compromising the accuracy and convergence speed.

REFERENCES

[1] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, “Termgrad: Ternary gradients to reduce communication in distributed deep learning,” in Advances in Neural Information Processing Systems, 2017.
[2] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, “Qsgd: Communication-efficient sgd via gradient quantization and encoding,” in Advances in Neural Information Processing Systems, 2017.
[3] A. F. Aji and K. Heafield, “Sparse communication for distributed gradient descent,” arXiv preprint arXiv:1704.05021, 2017.
[4] T. Alabi, J. D. Blanchard, B. Gordon, and R. Steinbach, “Fast k-selection algorithms for graphics processing units,” Journal of Experimental Algorithmics (JEA), vol. 17, pp. 4–2, 2012.
[5] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, p. 436, 2015.
[6] S. Han, H. Mao, and W. J. Dally, “Deep compression: Reducing the communication bandwidth for distributed deep neural networks,” in Advances in neural information processing systems, 2015, pp. 2674–2682.
[7] L. Wang, J. Ye, Y. Zhao, H. Li, S. Li, and Z. Xu, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.
[8] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A.Senior, P. Tucker, K. Yang, Q. V. Le et al., “Large scale distributed deep networks,” in Advances in neural information processing systems, 2012.
[36] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Advances in neural information processing systems, 2015.

[37] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, “Learning compact recurrent neural networks with block-term tensor decomposition,” arXiv preprint arXiv:1712.05134, 2017.
Lemma 6.1: Assume \( \eta_t \leq \frac{1}{2\sigma^2}, \theta_t^2 \leq \frac{1}{4} \). Then
\[
\frac{\eta_t}{4} \mathbb{E}[\|\nabla f(x^t)\|^2] \leq \mathbb{E}[f(x^t)] - \mathbb{E}[f(x^{t+1})] + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t}.
\tag{8}
\]

Proof. By Lipschitz continuity,
\[
f(x^{t+1}) = f(x^t - \eta_t \hat{v}_t)
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t \hat{v}_t \rangle + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t \hat{v}_t \rangle + \eta_t \|\nabla f(x^t)\| \|\hat{v}_t - v_t\| + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t \hat{v}_t \rangle + \eta_t \|\nabla f(x^t)\| \|\hat{v}_t - v_t\| + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} \|\hat{v}_t - v_t\|^2 + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t \theta_t^2}{2} \|v_t\|^2 + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\leq f(x^t) + \langle \nabla f(x^t), -\eta_t v_t \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t \theta_t^2}{2} \|v_t\|^2 + \frac{L}{2} \|\eta_t \hat{v}_t\|^2
\]
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
\[
\mathbb{E}[\|v_t\|^2|x^t] = \mathbb{E}[\|v_t - \nabla f(x^t)\|^2|x^t] + \|\mathbb{E}[v_t|x^t]\|^2
= \mathbb{E}[\|v_t - \nabla f(x^t)\|^2|x^t] + \|\nabla f(x^t)\|^2
\leq \frac{\sigma^2}{b_t} + \|\nabla f(x^t)\|^2
\]
We have
\[
\mathbb{E}[f(x^{t+1})|x^t] \leq f(x^t) + \langle \nabla f(x^t), -\eta_t \mathbb{E}[v_t|x^t] \rangle + \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} (L\eta_t + \theta_t^2) \mathbb{E}[\|v_t\|^2|x^t]
\leq f(x^t) - \frac{\eta_t}{2} \|\nabla f(x^t)\|^2 + \frac{\eta_t}{2} (L\eta_t + \theta_t^2) \left( \frac{\sigma^2}{b_t} + \|\nabla f(x^t)\|^2 \right)
= f(x^t) - \frac{\eta_t}{2} (1 - L\eta_t - \theta_t^2) \|\nabla f(x^t)\|^2 + (L\eta_t + \theta_t^2) \frac{\eta_t \sigma^2}{2b_t}
\]
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. \( \square \)