Fine-tuning Language Models over Slow Networks using Activation Compression with Guarantees

Anonymous Author(s)
Affiliation
Address
email

Abstract

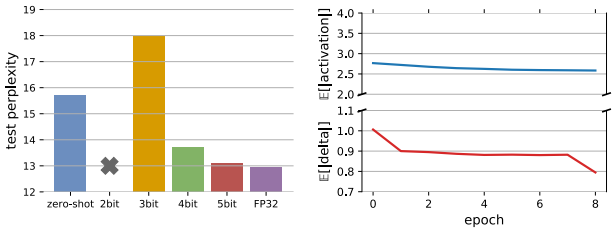
Communication compression is a crucial technique for modern distributed learning systems to alleviate their communication bottlenecks over slower networks. Despite recent intensive studies of gradient compression for data parallel-style training, compressing the *activations* for models trained with pipeline parallelism is still an open problem. In this paper, we propose AC-SGD, a novel activation compression algorithm for communication-efficient pipeline parallelism training over slow networks. Different from previous efforts in activation compression, instead of compressing activation values directly, AC-SGD compresses the *changes of the activations*. This allows us to show, to the best of our knowledge for the first time, that one can still achieve $O(1/\sqrt{T})$ convergence rate for non-convex objectives under activation compression, without making assumptions on gradient unbiasedness that do not hold for deep learning models with non-linear activation functions. We then show that AC-SGD can be optimized and implemented efficiently, without additional end-to-end runtime overhead. We evaluated AC-SGD to fine-tune language models with up to 1.5 billion parameters, compressing activations to 2-4 bits. AC-SGD provides up to $4.3\times$ end-to-end speed-up in slower networks, without sacrificing model quality. Moreover, we also show that AC-SGD can be combined with state-of-the-art gradient compression algorithms to enable "end-to-end communication compression": *All communications between machines, including model gradients, forward activations, and backward gradients are compressed into lower precision*. This provides up to $4.9\times$ end-to-end speed-up, without sacrificing model quality.

1 Introduction

Recent efforts in improving communication efficiency for distributed learning have significantly decreased the dependency of training deep learning models on fast data center networks — the *gradient* can be compressed to lower precision or sparsified [1, 2, 3, 4], which speeds up training over low bandwidth networks, whereas the *communication topology* can be decentralized [5, 6, 7, 8, 9, 10], which speeds up training over high latency networks. Indeed, today's state-of-the-art training systems, such as Pytorch [11, 12], Horovod [13], Bagua [14], and BytePS [15], already support many of these communication-efficient training paradigms.

However, with the rise of large foundation models [16] (e.g., BERT [17], GPT-3 [18], and CLIP[19]), improving communication efficiency via compression becomes more challenging. Current training systems for foundation models such as Megatron [20], Deepspeed [21], and Fairscale [22], allocate different layers of the model onto multiple devices and need to communicate — *in addition to* the gradients on the models — the *activations* during the forward pass and the *gradients on the activations* during the backward pass. Compressing these *activations* leads to a very different behavior compared

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.



(a) Fine-tune WikiText2 (b) Activation and delta

Figure 1: (a) Fine-tuning GPT2-1.5B with different activation precisions in communication; (b) Average absolute value of activations and their changes for GPT2-1.5B during training.

Table 1: Summary of technical results. AC-GC [25] and TinyScript [26] assume that the returned gradient is unbiased, whereas `AC-SGD` algorithm does not rely on such an assumption.

Algorithm	Assumptions on Quant. Grad.	Conv. Rate
SGD [27]	N/A	$\mathcal{O}(1/\sqrt{T})$
AC-GC [25]	Unbiased	$\mathcal{O}(1/\sqrt{T})$
TinyScript [26]	Unbiased	$\mathcal{O}(1/\sqrt{T})$
`AC-SGD`	N/A	$\mathcal{O}(1/\sqrt{T})$

with compressing the gradient — *simply compressing these activations in a stochastically unbiased way will lead to biases in the gradient that cannot be measured easily or expressed in closed form*. This either breaks the unbiasedness assumptions made by most gradient compression results [1, 2, 3, 4] or makes error compensation over gradient biases [23, 24] difficult to adopt.

Previous efforts on activation compression [28, 29, 30, 31, 32] illustrate, albeit mostly empirically, that large deep learning models can tolerate some compression errors on these activation values. However, when it comes to the underlying theoretical analysis, these efforts mostly make assumptions that do not apply to neural networks with non-linear activation functions — the only two recent efforts that claim theoretical convergence analysis [25, 26] assume that an unbiased compression on activations leads to an unbiased error on the gradient. Not surprisingly, these algorithms lead to suboptimal quality under relatively aggressive compression, illustrated in Figure 1a — in many cases, using activation compression to fine-tune a model might be worse than zero-shot learning without any fine-tuning at all.

In this paper, we focus on the problem of activation compression for training language models over slow networks by asking the following:

- **Q1.** *Can we design an algorithm for activation compression with rigorous theoretical guarantees on SGD convergence?*
- **Q2.** *Can such an algorithm be implemented efficiently without additional runtime overhead and outperform today's activation compression algorithms without accuracy loss?*

Our answers to both questions are ***Yes***. **(Contribution 1)** We propose `AC-SGD`, a novel algorithm for activation compression. The idea of `AC-SGD` is simple — instead of directly compressing the activations, *compress the change of activations for the same training example across epochs*. Intuitively, we expect `AC-SGD` to outperform simply compressing the activations because it enables an interesting "self-enforcing" dynamics: *the more training stabilizes $\rightarrow$ the smaller the changes of the model across epochs $\rightarrow$ the smaller the changes of activations for the same training example across epochs $\rightarrow$ the smaller the compression error using the same #bits $\rightarrow$ training stabilizes more.* **(Contribution 2)** The theoretical analysis of `AC-SGD` is non-trivial since we have to analyze the above dynamics and connect it to SGD convergence, which is quite different from most of today's results on gradient compression and error compensation. Under mild technical conditions and quantization functions with bounded error, we show that `AC-SGD` converges with a rate of $O(1/\sqrt{T})$ for non-convex objectives, the same as vanilla SGD [27, 33]. To the best of our knowledge, `AC-SGD` is the first activation compression algorithm with rigorous theoretical analysis that shows a convergence rate of $O(1/\sqrt{T})$ (without relying on assumptions of unbiased gradient). **(Contribution 3)** We then show that `AC-SGD` can be optimized and implemented efficiently, without adding additional end-to-end runtime overhead over non-compression and other compression schemes (it does require us to utilize more memory and SSD for storage of activations). **(Contribution 4)** We then conduct extensive experiments on sequence classification and language modeling datasets using DeBERTa-1.5B and GPT2-1.5B models, respectively. We show that `AC-SGD` can aggressively compress activations to 2-4 bits without sacrificing convergence performance, where direct quantization of activations fails to

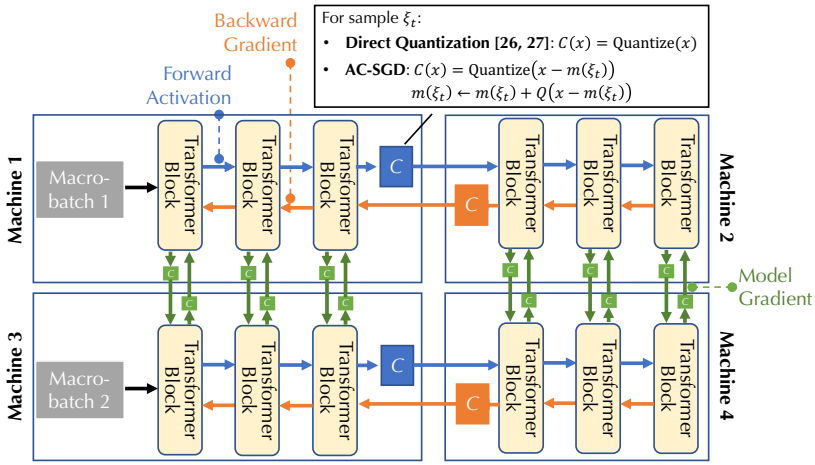


Figure 2: The communication pattern of training large language models with both data parallelism and pipeline model parallelism. C denotes a compression module. The goal of this paper is to understand the design of C for *forward activation* and *backward gradient*.

converge; in slow networks, `AC-SGD` achieves up to $4.3\times$ end-to-end speedup. **(Contribution 5)** Last but not least, we also show that `AC-SGD` can be combined with state-of-the-art gradient compression algorithms to enable "end-to-end communication compression": *All data exchanges between machines, including model gradients, forward activations, and backward gradients are compressed into lower precision*. This provides up to $4.9\times$ end-to-end speed-up, without sacrificing model quality.

2 Overview and Problem Formulation

Training large language models over multiple devices is a challenging task. Because of the vast number of parameters of the model and data examples, state-of-the-art systems need to combine different forms of parallelism. Figure 2 illustrates an example in which such a model is distributed over four machines: *(Pipeline Model Parallelism)* The model is partitioned onto Machine 1 and Machine 2 (similarly, Machine 3 and Machine 4), each of which is holding a subset of *layers*. To compute the gradient over the model using backpropagation, these two machines need to communicate the *activations* during the forward pass and the *gradient on activations* during the backward pass. *(Data Parallelism)* Machine 1 and Machine 3 (similarly, Machine 2 and Machine 4) process the same set of *layers* for different *macro-batches*. In this case, each of them will hold a replica of the same model. After the gradient over their model parameters are ready, they need to communicate the *model gradient*, usually by taking an average [11, 13, 14].

Communication Compression for Forward Activations and Backward Gradients. In slow networks, the communication among all machines often becomes the bottleneck [33]. To improve the speed of training, one can conduct *lossy compression* of the data before they are communicated, illustrated as the C module in Figure 2. When the model fits into a single machine, there have been intensive efforts on compressing the model gradient [1, 2, 3, 4]. However, when it comes to pipeline model parallelism, such compression techniques are less studied. In this paper, we focus on designing efficient communication compression algorithms to compress both forward activations and backward gradients. As we will show later, both can be compressed significantly with `AC-SGD` without hurting the model quality. We also show that it is possible to combine `AC-SGD` with state-of-the-art gradient compression techniques to enable the end-to-end compression scheme illustrated in Figure 2.

Problem Formulation. In this paper, we focus on the following technical problem. Note that, for the simplicity of notations, we present here the case where the model is partitioned onto $K=2$ machines. `AC-SGD` works for cases with $K>2$: (1) in experiments, we consider $K=8$, i.e., a single model is partitioned onto 8 machines; (2) in the supplementary material we provide the theoretical analysis for $K>2$.

Given a distribution of samples $\mathcal{D}$, we consider the following optimization task:

$$\min_{x\in\mathbb{R}^d} \quad f(x) := \mathbb{E}_{\xi\sim\mathcal{D}} F(b(a(\xi, x^{(a)}), x^{(b)})), \tag{2.1}$$

Algorithm 1 `AC-SGD` Algorithm

Initialize: x_0, learning rate γ, sub-network $a(-)$ weights $x^{(a)}$, sub-network $b(-)$ weights $x^{(b)}$, quantization function Q, array of previous messages m initialized to 0
for t = 1, . . . , T **do**
  Randomly sample ξ_t
  if ξ_t **not** seen before **then**
    Set $m(\xi_t) = a(\xi_t, x_t^{(a)})$
  else
    Update $m(\xi_t) \leftarrow m(\xi_t) + Q\big(a(\xi_t, x_t^{(a)}) - m(\xi_t)\big)$
  end if
  // Machine a sends $Q\big(a(\xi_t, x_t^{(a)}) - m(\xi_t)\big)$ to Machine b, which knows $m(\xi_t)$ through a local version of m
  Update $x_{t+1}^{(b)} \leftarrow x_t^{(b)} - \gamma \cdot \nabla_{x^{(b)}} (f \circ b)|_m$
  // Machine b sends $Q(\nabla_a (f \circ b)|_m)$ to Machine a
  Update $x_{t+1}^{(a)} \leftarrow x_t^{(a)} - \gamma \cdot Q(\nabla_a (f \circ b)|_m) \cdot \nabla_{x^{(a)}} a$
end for
Output: $x = (x_T^{(a)}, x_T^{(b)})$

where F is a loss function, $a(-)$ and $b(-)$ correspond to two sets of *layers* of the model — $a(-)$ has model $x^{(a)}$ and $b(-)$ has model $x^{(b)}$. In Figure 2, Machine 1 would hold $x^{(a)}$ and Machine 2 would hold $x^{(b)}$. In the following, we call the machine that holds $x^{(a)}$ Machine a and the machine that holds $x^{(b)}$ Machine b . In the standard backpropagation algorithm, the communication between these two machines are as follows:

- Given a data sample ξ, Machine a sends to Machine b the forward activation: $a(\xi, x^{(a)})$
- Machine b sends to Machine a the backward gradient on $a(\xi, x^{(a)})$.

Difficulties in Direct Quantization. A natural way at compressing forward activations is to send, instead of $a(\xi, x^{(a)})$, a quantized version $m(\xi, x^{(a)}) = Q(a(\xi, x^{(a)}))$. This is the quantization scheme that state-of-the-art methods such as AC-GC [25] and TinyScript [26] use. Both AC-GC [25] and TinyScript [26] assume that gradient is unbiased when $m(\xi, x^{(a)})$ is an unbiased estimator of $a(\xi, x^{(a)})$. This enables their convergence rates of $\mathcal{O}(1/\sqrt{T})$. However, because of the non-linearity of F and b in a deep learning model with non-linear activation functions, an unbiased $m(\xi, x^{(a)})$ *will* lead to biases on the gradient. In Appendix, we will provide an example showing that such a gradient bias will hurt SGD convergence even for a very simple optimization problem. On the theory side, previous efforts on understanding gradient bias [34] also has shown that even bounded bias on gradient can impact the converges of SGD. As we will show later, empirically, this bias can indeed lead to suboptimal models under aggressive compression.

Notation. Throughout the paper we use the following definitions:

- f^* is the optimal value of 2.1.
- N is the number of samples.
- $x_t = (x_t^{(a)}, x_t^{(b)})$ is the full model at iteration t.
- $\nabla f(\cdot)$ is the gradient of function f.
- $g_{\xi_t}(x_t) = \nabla F(\xi_t; x_t)$ is the stochastic gradient.
- $Q(\cdot)$ is the quantization function used to compress activations.
- $m(\cdot)$ is the message exchanged between a and b in the feed forward path.
- $\|\cdot\|$ denotes the L_2-norm.

3 `AC-SGD`: Theoretical Analysis and System Implementations

In this section we present the `AC-SGD`, with the goal to mitigate the above mentioned difficulties that appear in direct quantization of the activation functions.

3.1 AC-SGD Algorithm

Algorithm 1 illustrates the AC-SGD algorithm. The idea behind Algorithm 1 is simple — *instead of compressing the activations directly, compress the changes of activations for the same training example across epochs*. As illustrated in Algorithm 1, for iteration t and the data sample ξ_t, *if* it is the first time that ξ_t is sampled, Machine a communicates the full precision activations without any compression: $m(\xi_t) = a(\xi_t, x_t^{(a)})$ (Lines 4-5). Both machines will save $m(\xi_t)$ in a local buffer. If ξ_t has been sampled in previous iterations, Machine a communicates a compressed version:

$$Q(a(\xi_t, x_t^{(a)}) - m(\xi_t)),$$

where $m(\xi_t)$ was the previous message, stored in the local buffer. Both machines then update this local buffer:

$$m(\xi_t) \leftarrow m(\xi_t) + Q(a(\xi_t, x_t^{(a)}) - m(\xi_t)).$$

Machine b then use $m(\xi_t)$ as the forward activations, compute backward gradients, and communicate a quantized version of the backward gradient to Machine a (Line 11). We use

$$\delta_\xi = a(\xi_t, x_t^{(a)}) - m(\xi_t)$$

to denote the *message error* in sending the activations.

Update Rules. The above algorithm corresponds to the following update rules, at iteration t with sample ξ_t:

$$x_{t+1}^{(a)} = x_t^{(a)} - \gamma \cdot Q(\nabla_a (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})}) \cdot \nabla_{x^{(a)}} a|_{x_t^{(a)}},$$
$$x_{t+1}^{(b)} = x_t^{(b)} - \gamma \cdot \nabla_{x^{(b)}} (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})},$$

where γ is the learning rate, $\nabla_{x^{(b)}} (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})}$ is the gradient on $x^{(b)}$ using the compressed forward activations $(m(\xi, x_t^{(a)})$, and $Q(\nabla_a (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})})$ is the quantized backward gradient.

Setting $x_t = (x_t^{(a)}, x_t^{(b)})$, we can rephrase the update rule as

$$x_{t+1} = x_t - \gamma \cdot (g_\xi(x_t) + \Delta_\xi(x_t)),$$

where $g_\xi(x_t)$ is the stochastic gradient and $\Delta_\xi(x_t)$ is the *gradient error* introduced by communication compression. We have $\Delta_\xi = (\Delta_\xi^{(a)} + \Delta_\xi^{(Q)}, \Delta_\xi^{(b)})$ given by:

$$\Delta_\xi^{(Q)}(x_t) = Q(\nabla_a (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})}) \cdot \nabla_{x^{(a)}} a|_{x_t^{(a)}} - \nabla_a (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})} \cdot \nabla_{x^{(a)}} a|_{x_t^{(a)}},$$
$$\Delta_\xi^{(a)}(x_t) = \nabla_a (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})} \cdot \nabla_{x^{(a)}} a|_{x_t^{(a)}} - \nabla_a (f \circ b \circ a)|_{(x_t^{(a)}, x_t^{(b)})}$$
$$\Delta_\xi^{(b)}(x_t) = \nabla_{x^{(b)}} (f \circ b)|_{(m(\xi, x_t^{(a)}), x_t^{(b)})} - \nabla_{x^{(b)}} (f \circ b)|_{(a(\xi, x_t^{(a)}), x_t^{(b)})},$$

where $\Delta_\xi^{(Q)}(x_t)$ is the error introduced by the gradient quantization in the backpropagation part, whilst $\Delta_\xi^{(a)}(x_t)$ and $\Delta_\xi^{(b)}(x_t)$ are the errors that the gradients of a and b, respectively, inherit from the bias introduced in the forward pass.

3.2 Theoretical Analysis

We now prove the main theorem which states that, under some standard assumptions that are often used in the literature [27, 33], the convergence rate of AC-SGD algorithm is $O(1/\sqrt{T})$ for non-convex objectives, same as vanilla SGD.

Assumptions. We make several assumptions on the networks and the quantization. It is important to note is that we put no restrictions on either the message error δ_ξ, nor the gradient error Δ_ξ.

- **(A1: Lipschitz assumptions)** We assume that ∇f, $\nabla(f \circ b)$ and a are L_f, $L_{f \circ b}$-, and ℓ_a-Lipschitz, respectively, recalling that a function g is L_g-Lipschitz if
$$\|g(x) - g(y)\| \leq L_g \|x - y\|, \quad \forall x, \forall y.$$
Furthermore, we assume that a and $f \circ b$ have gradients bounded by C_a and $C_{f \circ b}$, respectively, i.e. $\|\nabla a(x)\| \leq C_a$, and $\|\nabla(f \circ b)(x)\| \leq C_{f \circ b}$.
- **(A2: SGD assumptions)** We assume that the stochastic gradient g_ξ is unbiased, i.e. $\mathbb{E}_\xi[g_\xi(x)] = \nabla f(x)$, for all x, and with bounded variance, i.e. $\mathbb{E}_\xi \|g_\xi(x) - \nabla f(x)\|^2 \leq \sigma^2$, for all x.

Theorem 3.1. *Suppose that Assumptions A1, A2 hold, and consider an unbiased quantization function $Q(x)$ which satisfies that there exists $c_Q < \sqrt{1/2}$ such that $\mathbb{E}\|x - Q(x)\| \leq c_Q\|x\|$, for all x.*[1] *Let $\gamma = \frac{1}{3(C+L_f)\sqrt{T}}$ be the learning rate, where*

$$C = \frac{4c_Q\ell_a(1+C_a)L_{fob}N}{\sqrt{1-2c_Q^2}}.$$

Then after performing T updates one has

$$\frac{1}{T}\sum_{t\in[T]}\mathbb{E}\|\nabla f(x_t)\|^2 \lesssim \frac{(C+L_f)(f(x_1)-f^*)}{\sqrt{T}} + \frac{\sigma^2+(c_Q C_a C_{fob})^2}{\sqrt{T}}. \tag{3.1}$$

We present the full proof of Theorem 3.1 in Appendix A, whereas here we explain the main intuition. The usual starting point in examining convergence rates is to use the fact that f has L_f-Lipschitz gradient. It is well known that this implies

$$\gamma\langle\nabla f(x_t), g_{\xi_t}(x_t)\rangle + f(x_{t+1}) - f(x_t) \leq -\langle\nabla f(x_t), \Delta_{\xi_t}(x_t)\rangle + \frac{\gamma^2 L_f}{2}\|g_{\xi_t}(x_t) + \Delta_{\xi_t}(x_t)\|^2.$$

After taking the expectation over all ξ_t and summing over all $t = 1,\ldots,T$, we easily see that the key quantity to bound is $\sum_{t=1}^{T}\mathbb{E}\|\tilde{\Delta}_{\xi_t}(x_t)\|^2$, where $\tilde{\Delta}_{\xi_t}(x_t) = (\Delta_{\xi_t}^{(a)}(x_t), \Delta_{\xi_t}^{(b)}(x_t))$. On the other hand, the main object that we can control is the message error, δ_ξ. Therefore, we first prove an auxiliary result which shows that $\|\tilde{\Delta}_{\xi_t}(x_t)\| \leq (1+C_a)\ell_a\|\delta_{\xi_t}(x_t)\|$, for all t. The key arguments for bounding δ_{ξ_t} closely follow the self-improving loop described in the introduction, and can be summarized as follows. Since we are compressing the information in such a way that we compare with all the accumulated differences, this allows us to unwrap the changes which appeared since the last time that we observed the current sample, in an iterative way. However, since these are gradient updates, they are bounded by the learning rate — as long as we have a quantization method that keeps enough information about the signal, we can recursively build enough saving throughout the process. In particular, the more stability we have in the process, the smaller the changes of the model and the compression error gets, further strengthening the stability.

Tightness. The bound is tight with respect to quantization — setting $c_Q = 0$ (implying $C = 0$), i.e. quantization does not incur any loss, recovers the original original SGD convergence (cf. [27, 33]).

3.3 System Implementations and Optimizations.

Additional storage and communication. `AC-SGD` requires us to store, for each data example ξ, the compressed activation $m(\xi)$ in a local buffer in memory or SSD. For example, in GPT2-XL training, a simple calculation shows that we need an approximately extra 1TB storage. When using data parallelism, it reduces to 1TB / # parallel degree, but also incurs communication overhead if data is shuffled in every epoch. In addition, when the example ξ is sampled again, $m(\xi)$ needs to be (1) loaded from this local buffer to the GPU, and (2) updated when a new value for $m(\xi)$ is ready.

Optimization. It is easy to implement and optimize the system such that this additional loading and updating step do not incur significant overhead on the end-to-end runtime. This is because of the fact that the GPU computation time for a forward pass is usually much longer than the data transfer time to load the activations — for GPT2-XL with 1.5 billion model parameters, a single forward pass on 6 layers require 44 ms, whereas loading $m(\xi)$ need 0.2 ms from memory and 12 ms from SSD. One can simply pre-fetch $m(\xi)$ right before the forward pass, and hide it within the forward pass of other data examples. Similarly, updating $m(\xi)$ can also be hidden in the backward computation. It is also simple to reduce the communication overhead by shuffling data only once or less frequently.

[1] Even for a very simple quantization function $Q(x) = \|x\| \cdot \lceil x/||x|| \rfloor$, where $\lceil\cdot\rfloor$ denotes rounding to the closest $k/2^b$, stochastically, through a simple bound $c_Q = \sqrt{d}/2^b$, 6 bits suffice in a low-dimensional ($\sim 10^3$), 11 bits in a high-dimensional scenario ($\sim 10^6$), and 16 bits in a super-high-dimensional scenario ($\sim 10^9$). In practice, as we show in the experiments, we observe that for 2-4 bits are often enough for fine-tuning GPT-2 style model. This leaves interesting direction for future exploration as we expect a careful analysis of sparsity together with more advanced quantization functions can make this condition much weaker.

4 Evaluation

We demonstrate that AC-SGD can significantly speed up fine-tuning large language models in slow networks. Specifically, we show: (1) on four standard benchmark tasks, AC-SGD can tolerate aggressive compression on the activations (2-4 bits) and backward gradients (4-8 bits), without hurting convergence and final loss, whereas direct quantization converges to a worse loss or even diverge, (2) in slow networks, AC-SGD provide an end-to-end speed-up up to $4.3\times$, and (3) AC-SGD can be combined with state-of-the-art gradient compression methods and achieve an end-to-end speed-up of up to $4.9\times$.

4.1 Experimental Setup

Datasets and Benchmarks. We consider both *sequence classification* and *language modeling* tasks with state-of-the-art foundation model. For the sequence classification task, we fine-tune a 1.5 billion parameter DeBERTa model[2] on two datasets: QNLI and CoLA. For the language modeling task, we fine-tune the GPT2 model with 1.5 billion parameters[3] on two datasets: WikiText2 and arXiv abstracts. All datasets are publicly available and do not contain sensitive or offensive content. Detailed setup can be found in Appendix B.

Distributed Cluster. We conduct our experiments on AWS with 8-32 `p3.2xlarge` instances, each containing a V100 GPU. For a single pipeline, we partition a model onto 8 machines. When combined with data parallelism, we use 32 instances — data parallel degree is 4 and pipeline parallel degree is 8. By default, instances are interconnected with 10Gbps bandwidth. We simulate slow networks by controlling the communication bandwidth between instances using Linux traffic control.

Baselines. We compare with several strong baselines:

1. `FP32`: in which all communications are in 32 bit floating point without any compression.
2. `DirectQ` [25, 26]: in which activations and backward gradients are directly compressed.

We use a simple, uniform compression scheme, which first normalizes a given vector into $[-1,1]$ and quantize each number into a b-bit integers by uniforming partitioning the range $[-1,1]$ into 2^b intervals [31]. Additional details of the configuration can be found in Appendix C.

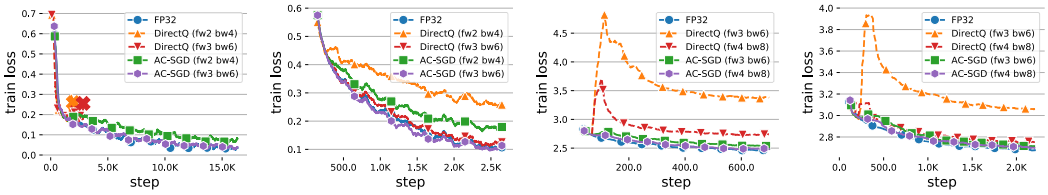
Hyperparameter Tuning. We conduct careful tuning for all methods on all datasets. We perform grid search to choose learning rate from {2.5e-6, 3e-6, 5e-6, 1e-5} and macro-batch size from {32, 64, 96} for best model performance.

4.2 Results

Convergence. We first compare the convergence behavior of different methods. For all compression methods, we try various settings: `fw`x `bw`y means that we use x bits for forward activation and y bits for backward gradients. Figure 3 shows the convergence behavior for the sequence classification and language modeling tasks. `FP32` converges fast since it does not introduce any compression errors. `DirectQ`, under aggressive compression, can converge to a significantly worse model, or even diverge. This is not surprising, given the biases on model gradients that direct quantization introduced. On the other hand, AC-SGD converges almost as fast as `FP32` in terms of number of training steps.

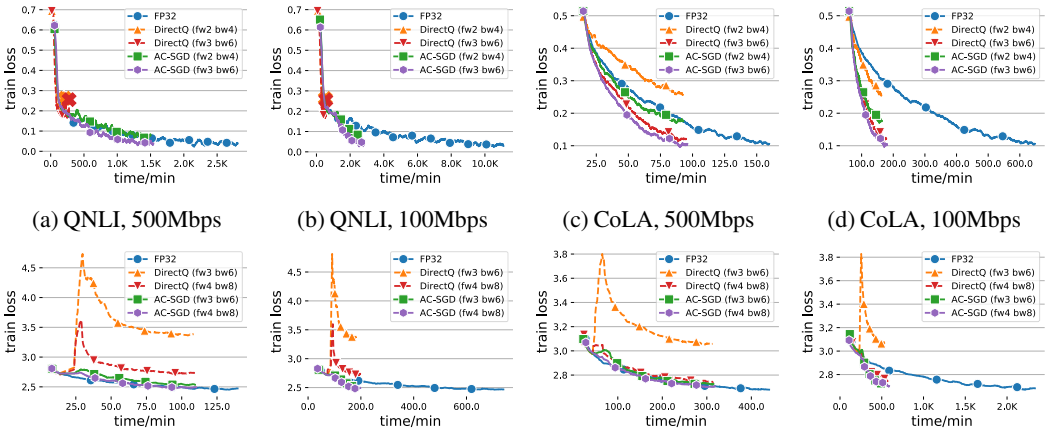
End-to-End Runtime. We show the end-to-end runtime of different methods under slow networks. As illustrated in Figure 4, AC-SGD achieves a $4.3\times$ end-to-end speed-up comparing with that of `FP32` (in terms of time to the same loss), illustrating the importance of communication compression in slow networks. Table 2 shows the training throughput. Another interesting observation is that when the network is $100\times$ slower (from 10Gbps to 100Mbps), the training is only about $1.28\times - 1.14\times$ slower! This is exciting — if AC-SGD were to be deployed in a in real-world geo-distributed decentralized networks, the training throughput of these fine-tuning tasks would be almost as fast as the performance inside a data center!

[2] we use the v2-xxlarge checkpoint: https://huggingface.co/microsoft/deberta-v2-xxlarge.
[3] we use the extra large checkpoint: https://huggingface.co/gpt2-xl.



(a) QNLI, DeBERTa-1.5B (b) CoLA, DeBERTa-1.5B (c) WikiText2, GPT2-1.5B (d) arXiv, GPT2-1.5B

Figure 3: Convergence (loss vs. # stpes) of different approaches. $\times$ represents divergence.



(a) QNLI, 500Mbps (b) QNLI, 100Mbps (c) CoLA, 500Mbps (d) CoLA, 100Mbps

(e) WikiText2, 500Mbps (f) WikiText2, 100Mbps (g) arXiv, 500Mbps (h) arXiv, 100Mbps

Figure 4: End-to-end training performance over different network configurations. $\times$ represents divergence.

Table 2: Training Throughput. We show DeBERTa and GPT2 on QNLI and WikiText2, other tasks are similar and we leave them to Appendix.

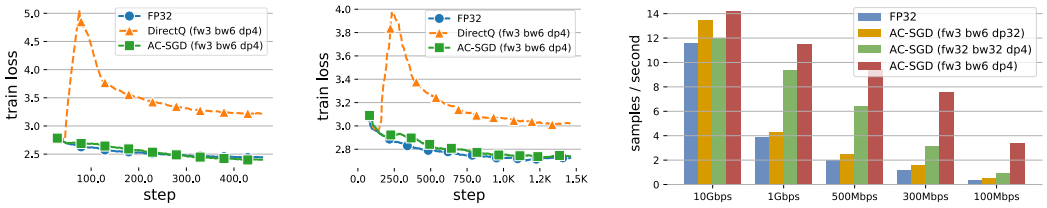
Network Bandwidth	DeBERTa-1.5B			GPT2-1.5B		
	FP32	`DirectQ` fw2 bw4 / fw3 bw6	`AC-SGD` fw2 bw4 / fw3 bw6	FP32	`DirectQ` fw3 bw6 / fw4 bw8	`AC-SGD` fw3 bw6 / fw4 bw8
10 Gbps	$12.9_{\pm 0.02}$	$13.6_{\pm 0.02}$ / $13.6_{\pm 0.02}$	$13.6_{\pm 0.02}$ / $13.5_{\pm 0.02}$	$3.8_{\pm 0.01}$	$4.0_{\pm 0.01}$ / $4.1_{\pm 0.01}$	$4.0_{\pm 0.01}$ / $4.0_{\pm 0.01}$
1 Gbps	$9.6_{\pm 0.02}$	$13.3_{\pm 0.02}$ / $13.1_{\pm 0.02}$	$13.3_{\pm 0.02}$ / $13.0_{\pm 0.02}$	$3.2_{\pm 0.01}$	$4.0_{\pm 0.01}$ / $4.0_{\pm 0.01}$	$4.0_{\pm 0.01}$ / $3.9_{\pm 0.01}$
500 Mbps	$6.2_{\pm 0.03}$	$13.0_{\pm 0.03}$ / $12.6_{\pm 0.03}$	$12.9_{\pm 0.03}$ / $12.5_{\pm 0.03}$	$2.7_{\pm 0.02}$	$3.9_{\pm 0.01}$ / $3.9_{\pm 0.01}$	$3.9_{\pm 0.01}$ / $3.9_{\pm 0.01}$
300 Mbps	$4.4_{\pm 0.04}$	$12.5_{\pm 0.02}$ / $11.9_{\pm 0.03}$	$12.4_{\pm 0.03}$ / $11.8_{\pm 0.03}$	$1.8_{\pm 0.02}$	$3.9_{\pm 0.01}$ / $3.8_{\pm 0.01}$	$3.8_{\pm 0.01}$ / $3.8_{\pm 0.01}$
100 Mbps	$1.6_{\pm 0.04}$	$10.7_{\pm 0.03}$ / $9.4_{\pm 0.03}$,	$10.6_{\pm 0.03}$ / $9.1_{\pm 0.03}$	$0.5_{\pm 0.02}$	$3.5_{\pm 0.02}$ / $3.0_{\pm 0.02}$	$3.4_{\pm 0.01}$ / $3.0_{\pm 0.02}$

Moreover, `AC-SGD` does not introduce significant runtime overhead over direct quantization. From Table 2, we see that `AC-SGD` is essentially as efficient as direct quantization compression in throughput.

4.3 End-to-end Communication Compression: `AC-SGD` + QuantizedAdam

`AC-SGD` can also be combined with existing methods on gradient compression. This allows us to compress *all* communications during training. We combine `AC-SGD` with QuantizedAdam [35], an error compensation-based gradient compression algorithm for data parallel training.

We compress the forward activations with 3 bits, the backward gradient with 6 bits, and model gradient with 4 bits. Figure 5 illustrates the convergence and the training throughput under different network configuration. We see that `AC-SGD` converges well when combined with QuantizedAdam (Figure 5(a, b)). On the other hand, `DirectQ`, when combined with gradient compression, converges to a much worse model. In terms of training throughput, with both activation and gradient compression, we can achieve up to 8.5$\times$ throughput improvement compared to the no-compression baseline (Figure 5(c)). We also see that *both* activation and gradient compression are important in terms of improving



(a) WikiText2, GPT2-1.5B (b) arXiv, GPT2-1.5B (c) Training Throughput

Figure 5: Convergence and Throughput of AC-SGD combined with gradient compression.

end-to-end training thoughput — as illustrated in Figure 5(c), disabling any of them will lead to a much lower training throughput.

5 Related Work

Distributed training of foundation models. Modern distributed training of deep neural networks goes beyond data parallelism [20, 36, 37, 37, 36] due to the advance of the large-scale foundation models [16], such as BERT [17], GPT-3 [18], and CLIP[19]. Popular systems to support foundation model training include Megatron [20], Deepspeed [21], and Fairscale [22]. To scale out the training of large-scale models, pipeline parallelism (e.g., Gpipe[36], Pipedream[38, 39]) is a popular option, where the model is partitioned into multiple stages, different stages are allocated on different GPUs and the exchange of activations and gradients on activations goes through network communication.

Communication compression of distributed learning. Communication compression is an effective system relaxation for distributed training, especially in data parallelism [40, 41, 42, 43, 44, 45, 46, 35, 47, 48, 49, 50]. Popular techniques include quantization [1, 2, 3, 4], sparsification [51, 52, 53, 54], sketching [55, 56] and error compensation [23]. Recently, TinyScript [26] proposes to compress activations and gradients simultaneously.

Sparse Learning for activation compression. Sparse learning [57, 58, 59, 60, 61, 62, 63, 64, 26] has become increasingly popular for training neural networks, as it can significantly reduce the use of computation and memory while preserving the generalization of such models. In particular, activation compression methods [28, 29, 30, 31, 32, 25] are proposed to reduce the memory footprint by adopting lossless [65, 66, 67] and lossy [68, 25, 69] compression techniques in the training of various deep neural networks (e.g., CNN[61, 70, 71], GNN[72]). These approaches usually compute the activation with *full precision* in forward propagation, adopt the compression method over the activation, and store the compressed version in DRAM for later use in backward propagation. Thus, compression does not introduce any error in forward propagation in contrast to the scenario of communicating compressed activation in the distributed setting.

Delta-based compression. Delta-based compression [73] is a classic solution to various system problems. Recent research has also used the property of spatial proximity within activation in neural network training based on empirical observations [74, 75, 69]. However, to our knowledge, no attempt has been made to consider the proximity of activation through training epochs to enable efficient compression with theoretical guarantee.

6 Conclusion

In this paper, we discuss how to adopt communication compression for activations in distributed learning. We proposed AC-SGD, a novel activation compression algorithm for communication-efficient pipeline parallelism training over slow networks. AC-SGD achieves $O(1/\sqrt{T})$ convergence rate for non-convex optimization without making assumptions on gradient unbiasedness. Our empirical study suggests that AC-SGD can achieve up to $4.3\times$ speedup for pipeline parallelism. When combined with gradient compression in data parallelism, the end-to-end speed-up can be up to $4.9\times$.

References

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *arXiv preprint arXiv:1610.02132*, 2016.

[2] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *International Conference on Machine Learning*, pages 4035–4043. PMLR, 2017.

[3] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.

[4] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1508–1518, 2017.

[5] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*, pages 3478–3487. PMLR, 2019.

[6] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-sgd: a decentralized pipelined sgd framework for distributed deep net training. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8056–8067, 2018.

[7] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5336–5346, 2017.

[8] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[9] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7663–7673, 2018.

[10] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856. PMLR, 2018.

[11] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12).

[12] Pytorch-lightning. `https://www.pytorchlightning.ai/`.

[13] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[14] Gan, Shaoduo and Lian, Xiangru and Wang, Rui and Chang, Jianbin and Liu, Chengjun and Shi, Hongmei and Zhang, Shengzhuo and Li, Xianghong and Sun, Tengxu and Jiang, Jiawei and others. BAGUA: scaling up distributed learning with system relaxations. *Proceedings of the VLDB Endowment*, 15(4):804–813, 2021.

[15] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous gpu/cpu clusters. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 463–479, 2020.

[16] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[20] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[21] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[22] Mandeep Baines, Shruti Bhosale, Vittorio Caggiano, Naman Goyal, Siddharth Goyal, Myle Ott, Benjamin Lefaudeux, Vitaliy Liptchinsky, Mike Rabbat, Sam Sheiffer, et al. Fairscale: A general purpose modular pytorch library for high performance and large scale training, 2021.

[23] Hanlin Tang, Chen Yu, Xiangru Lian, Tong Zhang, and Ji Liu. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *International Conference on Machine Learning*, pages 6155–6165. PMLR, 2019.

[24] Hanlin Tang, Xiangru Lian, Shuang Qiu, Lei Yuan, Ce Zhang, Tong Zhang, and Ji Liu. Deepsqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. *arXiv preprint arXiv:1907.07346*, 2019.

[25] R David Evans and Tor Aamodt. Ac-gc: Lossy activation compression with guaranteed convergence. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *International Conference on Machine Learning*, pages 3304–3314. PMLR, 2020.

[27] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[28] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2016.

[29] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[30] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789. IEEE, 2018.

[31] Ayan Chakrabarti and Benjamin Moseley. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems*, 32, 2019.

[32] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pages 1803–1813. PMLR, 2021.

[33] Ji Liu and Ce Zhang. Distributed learning systems with first-order methods. *arXiv preprint arXiv:2104.05245*, 2021.

[34] Ahmad Ajalloeian and Sebastian U. Stich. On the convergence of sgd with biased gradients, 2020.

[35] Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 1-bit adam: Communication efficient large-scale training with adam's convergence speed. In *International Conference on Machine Learning*, pages 10118–10129. PMLR, 2021.

[36] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[37] Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems*, 3:269–296, 2021.

[38] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[39] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pages 7937–7947. PMLR, 2021.

[40] Chia-Yu Chen, Jiamin Ni, Songtao Lu, Xiaodong Cui, Pin-Yu Chen, Xiao Sun, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Wei Zhang, et al. Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training. *Advances in Neural Information Processing Systems*, 33:13551–13563, 2020.

[41] Cong Xie, Shuai Zheng, Sanmi Koyejo, Indranil Gupta, Mu Li, and Haibin Lin. Cser: Communication-efficient sgd with error reset. *Advances in Neural Information Processing Systems*, 33:12593–12603, 2020.

[42] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

[43] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtarik. Acceleration for compressed gradient descent in distributed and federated optimization. In *International Conference on Machine Learning*, pages 5895–5904. PMLR, 2020.

[44] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pages 8253–8265. PMLR, 2020.

[45] Mher Safaryan and Peter Richtárik. Stochastic sign descent methods: New algorithms and better theory. In *International Conference on Machine Learning*, pages 9224–9234. PMLR, 2021.

[46] Eduard Gorbunov, Konstantin P Burlachenko, Zhize Li, and Peter Richtárik. Marina: Faster non-convex distributed learning with compression. In *International Conference on Machine Learning*, pages 3788–3798. PMLR, 2021.

[47] Mher Safaryan, Filip Hanzely, and Peter Richtárik. Smoothness matrices beat smoothness constants: better communication compression techniques for distributed optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

[48] Xun Qian, Peter Richtárik, and Tong Zhang. Error compensated distributed sgd can be accelerated. *Advances in Neural Information Processing Systems*, 34, 2021.

[49] Ahmed M Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, and Marco Canini. An efficient statistical-based gradient compression technique for distributed training systems. *Proceedings of Machine Learning and Systems*, 3:297–322, 2021.

[50] Hongyi Wang, Saurabh Agarwal, and Dimitris Papailiopoulos. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021.

[51] J Wangni, J Liu, J Wang, and T Zhang. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31:1299, 2018.

[52] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. The convergence of sparsified gradient methods. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5977–5987, 2018.

[53] Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. Atomo: communication-efficient learning via atomic sparsification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9872–9883, 2018.

[54] Jialei Wang, Mladen Kolar, Nathan Srebro, and Tong Zhang. Efficient distributed learning with sparsity. In *International Conference on Machine Learning*, pages 3636–3645. PMLR, 2017.

[55] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. Sketchml: Accelerating distributed machine learning with data sketches. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 1269–1284, 2018.

[56] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Ion Stoica, Raman Arora, et al. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*, pages 13144–13154, 2019.

[57] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[58] Md Aamir Raihan and Tor Aamodt. Sparse weight activation training. *Advances in Neural Information Processing Systems*, 33:15625–15638, 2020.

[59] Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

[60] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[61] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655. PMLR, 2019.

[62] LIU Junjie, XU Zhe, SHI Runbin, Ray CC Cheung, and Hayden KH So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2019.

[63] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

[64] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems*, 34, 2021.

[65] Esha Choukse, Michael B Sullivan, Mike O'Connor, Mattan Erez, Jeff Pool, David Nellans, and Stephen W Keckler. Buddy compression: Enabling larger memory for deep learning and hpc workloads on gpus. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 926–939. IEEE, 2020.

[66] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W Keckler. Compressing dma engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 78–91. IEEE, 2018.

[67] Alberto Delmás Lascorz, Sayeh Sharify, Isak Edo, Dylan Malone Stuart, Omar Mohamed Awad, Patrick Judd, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Zissis Poulos, et al. Shapeshifter: Enabling fine-grain data width adaptation in deep learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 28–41, 2019.

[68] Sian Jin, Guanpeng Li, Shuaiwen Leon Song, and Dingwen Tao. A novel memory-efficient deep learning training framework via error-bounded lossy compression. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 485–487, 2021.

[69] R David Evans, Lufei Liu, and Tor M Aamodt. Jpeg-act: accelerating deep learning via transform-based lossy compression. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 860–873. IEEE, 2020.

[70] Georgios Georgiadis. Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7085–7095, 2019.

[71] Denis Gudovskiy, Alec Hodgkinson, and Luca Rigazio. Dnn feature map compression using learned representation over gf (2). In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[72] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2021.

[73] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Michael A Kozuch, Phillip B Gibbons, and Todd C Mowry. Base-delta-immediate compression: Practical data compression for on-chip caches. In *2012 21st international conference on parallel architectures and compilation techniques (PACT)*, pages 377–388. IEEE, 2012.

[74] Omar Mohamed Awad, Mostafa Mahmoud, Isak Edo, Ali Hadi Zadeh, Ciaran Bannon, Anand Jayarajan, Gennady Pekhimenko, and Andreas Moshovos. Fpraker: A processing element for accelerating neural network training. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 857–869, 2021.

[75] Andrei Bersatti, Nima Shoghi Ghalehshahi, and Hyesoon Kim. Neural network weight compression with nnw-bdi. In *The International Symposium on Memory Systems*, pages 335–340, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [N/A]
- Did you include the license to the code and datasets? [N/A]
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] `AC-SGD` requires additional storage to store the activation for each data example. We describe the limitation from the perspectives of algorithm and implementation in Section 3.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Language models increase the inequality on employment as suggested by DeepMind. The acceleration of training may lead to more misuse of language models. Our work also make positive social impact in privacy protection because it can also be applied in split learning as shown in Appendix.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We repeated some of the experiments for at least three times and reported the average in the paper. Details are reported in Appendix.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Our system was built on PyTorch and CuPy and was evaluated using publicly available abstracts (QNLI, CoLA, WikiText2 and arXiv abstracts). All of them use open-source licenses and can be used for non-commercial educational purposes.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]