Journal of Parallel and Distributed Computing 185 (2024) 104811

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc

Sketch-fusion: A gradient compression method with multi-layer fusion for communication-efficient distributed training

Lingfei Dai [a,b], Luqi Gong [a], Zhulin An [a], Yongjun Xu [a], Boyu Diao [a,*]

[a] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[b] College of Computer Science, University of Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Keywords:
Gradient compression
Multi-layer fusion
Distributed stochastic gradient descent
Deep learning training

ABSTRACT

Gradient compression is an effective technique for improving the efficiency of distributed training. However, introducing gradient compression can reduce model accuracy and training efficiency. Furthermore, we also find that using a layer-wise gradient compression algorithm would lead to significant compression and communication overhead, which can negatively impact the scaling efficiency of the distributed training system. To address these issues, we propose a new method called $Sketch-Fusion$ SGD, which leverages the Count-Sketch data structure to enhance the scalability and training speed of distributed deep learning systems. Moreover, our method employs LayerFusion to optimize gradient compression algorithms' scalability and convergence efficiency by formulating an optimal multi-layer fusion strategy without introducing extra hyperparameters. We evaluate our method on a cluster of 16 GPUs and demonstrate that it can improve training efficiency by up to 18.6% without compromising the model's accuracy. In addition, we find that applying our LayerFusion algorithm to other gradient compression methods improved their scalability by up to 2.87×.

1. Introduction

Deep learning has gained widespread popularity in various fields such as Computer Vision, Natural Language Processing, and Speech Recognition [24,13,31]. Most deep learning networks have achieved state-of-the-art performance across different domains [43,53,36,22,57]. Stochastic gradient descent (SGD) and its various variants [12] remain the primary methods for training deep learning networks. However, deep learning models require large-scale datasets such as ImageNet [37] for training. With the rapid increase in the number of parameters in deep neural networks (DNNs), training times have become longer, sometimes taking days to weeks [15].

To enhance the training efficiency of deep learning models, high-performance processors such as NVIDIA A100 [9], Google TPU [21], and Huawei Ascend910 [51] are employed to reduce training times. Furthermore, synchronous data parallel stochastic gradient descent is a widely adopted approach for distributed learning [27]. During the communication phase of data parallel training, a significant number of gradients need to be transmitted between servers. The network communication overhead for parameter information poses a severe bottleneck to the performance of distributed training [26].

In recent years, there has been an increased focus on addressing communication bottlenecks in distributed training, leading to a lot of research. A significant body of work has proposed gradient compression algorithms [16,18,49,54] to reduce the amount of data transferred simultaneously, thereby reducing communication overhead and improving scalability. Two primary types of gradient compression methods have emerged: 1) sparsification, which selects a subset of gradients for aggregation [45,30,47]; and 2) quantization, which quantizes gradients (e.g., FP32) to fewer bits (e.g., INT8) [3,10]. Theoretically, gradient compression can significantly reduce communication overhead thanks to smaller communication data sizes [4,28].

Gradient compression reduces communication overhead in distributed training, but it can harm model accuracy and scalability [19,38,52,14]. To overcome this challenge, we propose a new method, $Sketch-Fusion$ SGD, which compresses gradients using the Count-Sketch data structure on local servers. We found that using a layer-wise gradient compression algorithm would lead to significant compression and communication overhead. Therefore, we also propose a multi-layer fusion algorithm, called LayerFusion, to optimize the scalability and convergence efficiency of gradient compression algorithms and accelerate the distributed training of DNNs.

* Corresponding author.
E-mail address: diaoboyu2012@ict.ac.cn (B. Diao).

https://doi.org/10.1016/j.jpdc.2023.104811
Received 12 March 2023; Received in revised form 8 July 2023; Accepted 19 November 2023
Available online 24 November 2023

Our contributions are summarized as follows:

- We introduce $Sketch - Fusion$ SGD, which can effectively improve the training speed of various DNNs and datasets under low-bandwidth networks.
- We propose LayerFusion as a heuristic algorithm to improve the scalability of gradient compression algorithms and guarantee their convergence rate.
- We demonstrate the effectiveness of our approach by applying LayerFusion to various gradient compression algorithms.

The rest of the paper is organized as follows. Section 2 provides background information on deep learning, stochastic gradient descent, and distributed learning. In section 3, we discuss the limitations of current gradient compression algorithms and the motivation for our proposed approach. Section 4 introduces our proposed $Sketch-Fusion$ SGD method with LayerFusion and provides a theoretical analysis of its convergence rate. Section 5 presents experimental results and discussions, including comparisons with other compression methods and analyses of the trade-offs between compression ratio, communication overhead, and model accuracy. Finally, we conclude the paper in Section 6 and suggest directions for future research.

2. Background

We first introduce the communication overhead in distributed training and then describe the methods to reduce it in the current DL framework.

2.1. Communication in distributed deep learning

The data-parallel paradigm of distributed deep learning (DDL) involves each accelerator (e.g., GPU, TPU) having a replica of the model, and the training dataset is divided into subsets based on the number of GPUs. Each iteration involves the accelerator reading a batch of training data from its own training data subset, calculating the gradient tensor by forward and backward propagation, and using it to update the DNN model. An important step is aggregating the gradient tensors computed by all GPUs synchronously or asynchronously. Synchronous data-parallel DDL, where all GPUs communicate the gradient tensors and wait for the aggregated results before the next iteration, is the de facto standard used by DDL frameworks [27,20,39]. Asynchronous data-parallel DDL, where GPUs do not wait for aggregation to complete, may harm model accuracy [8].

In distributed training clusters with multiple machines, there is both intra-machine communication and inter-machine communication. Hierarchical communication is often used in DDL frameworks [27,20,39], where gradient synchronization is carried out in three steps: 1) gradients are aggregated among GPUs within one machine; 2) they are then aggregated across machines; 3) the aggregated gradients are communicated within one machine again to ensure all GPUs have the same synchronized results. Some frameworks [27,39] also support flat communication, where all GPUs join the same collective operation and have only one communication step.

2.2. Gradient compression

Gradient compression (GC) algorithms have been proposed to reduce communication overhead in DDL. These algorithms can be broadly categorized into two types: sparsification and quantization. Sparsification involves selecting a subset of gradients for synchronization [30,2], which minimizes gradient exchange while ensuring model accuracy. Quantization, on the other hand, reduces communication overhead by reducing the precision of gradients. This is achieved by mapping FP32 gradients to lower bit precision, such as 8 bits [10], 2 bits [19], or even 1 bit [7]. These compression algorithms have been shown to preserve model convergence and impose negligible impact on model accuracy when combined with error-feedback mechanisms, as validated through theoretical proofs and empirical studies [50,44].

2.3. Computation and communication overlap

A deep neural network comprises multiple layers, and the back-propagation process involves layer-by-layer computation, meaning that the gradient of the previous layer is already available when calculating the gradient of the current layer. To enhance the scalability and performance of DNNs, many studies [55,40,27,45,35,39] have proposed overlapping computation and communication by transmitting partial gradient information as soon as it becomes available instead of waiting for all gradient information to be ready. This approach, known as wait-free back-propagation (WFBP), is now widely adopted in popular machine learning frameworks, such as PyTorch [27], TensorFlow [1], and Horovod [39], to boost the training efficiency of distributed DNNs. The concept of overlapping computation and communication is also exploited in our LayerFusion algorithm.

3. Motivation

In this section, we aim to assess the scalability and compression efficiency of existing gradient compression algorithms through detailed experimental evaluations. To analyze the effect of gradient compression, we start by examining gradient calculation. We then conduct large-scale experiments to investigate the scalability of the GC algorithms across a range of models. The experimental setup is consistent with that described in Section 5.

3.1. Performance verification

To evaluate the performance of gradient compression, we conducted experiments to evaluate several popular compression methods, including sparsification and quantization methods. The performance of a compression algorithm can be measured by its scaling factor [56] and the time required for compression and decompression. The scaling factor is defined as the ratio of the training time on n accelerators to the training time on a single accelerator, denoted as T_n and T_1, respectively. Specifically, the scaling factor is computed as:

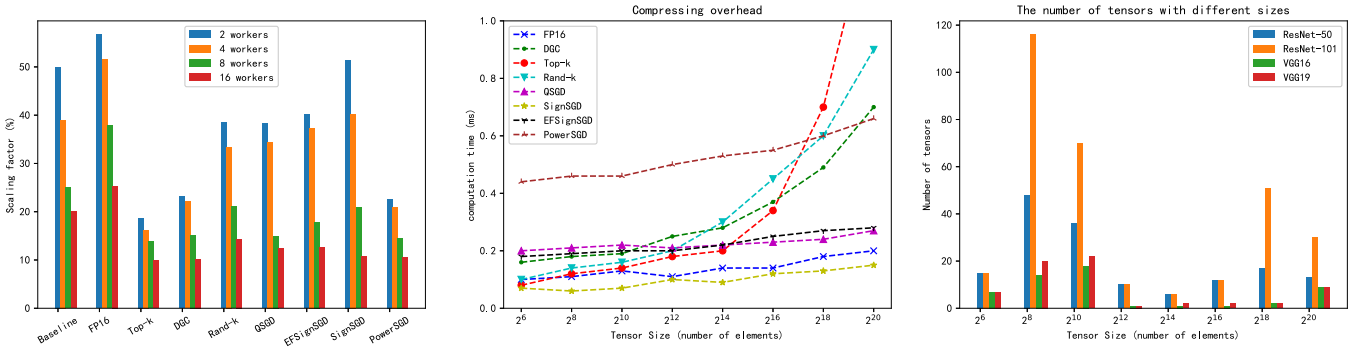
$$scaling\,factor = \frac{T_n}{nT_1} \tag{1}$$

Fig. 1a shows the scaling factors of various compression algorithms with layer-wise compression for ResNet-20 on the CIFAR10 dataset, with the baseline referring to full-precision gradient training without using the compression algorithm. Despite the good expansion efficiency of these compression algorithms in theory, actual experimental results show that they do not scale well and perform even worse than the baseline. In particular, the Top-k and DGC [30] algorithms have a performance more than 40% lower than the baseline.

In addition, we also measure the compression-decompression time of different gradient compression algorithms using ResNet-50 on V100 GPUs. Table 1 shows that most compression algorithms take more than 100 ms to compress and decompress gradients for ResNet-50 on V100 GPUs. The slowest algorithm is DGC, which takes 242 ms, while SignSGD takes only 13 ms.

3.2. Rethinking gradient compression

To analyze the poor performance of gradient compression, we conducted tests to measure the compressing overhead of different gradient compression algorithms across varying tensor sizes. The results were presented in Fig. 1b. From the graph, it is evident that the compression overhead of most algorithms cannot be ignored regardless of tensor size. During distributed training, deep neural network models have multiple



(a) The scaling factors of ResNet-20 on CIFAR10 dataset with various gradient compression algorithms.

(b) Compressing overhead

(c) The number of tensors with different sizes

Fig. 1. (a) The scaling factors of ResNet-20 on CIFAR10 dataset with various gradient compression algorithms. (b) The compressing overhead of tensors of different sizes under different compression algorithms. (c) The total number of tensors for synchronization in different DNNs.

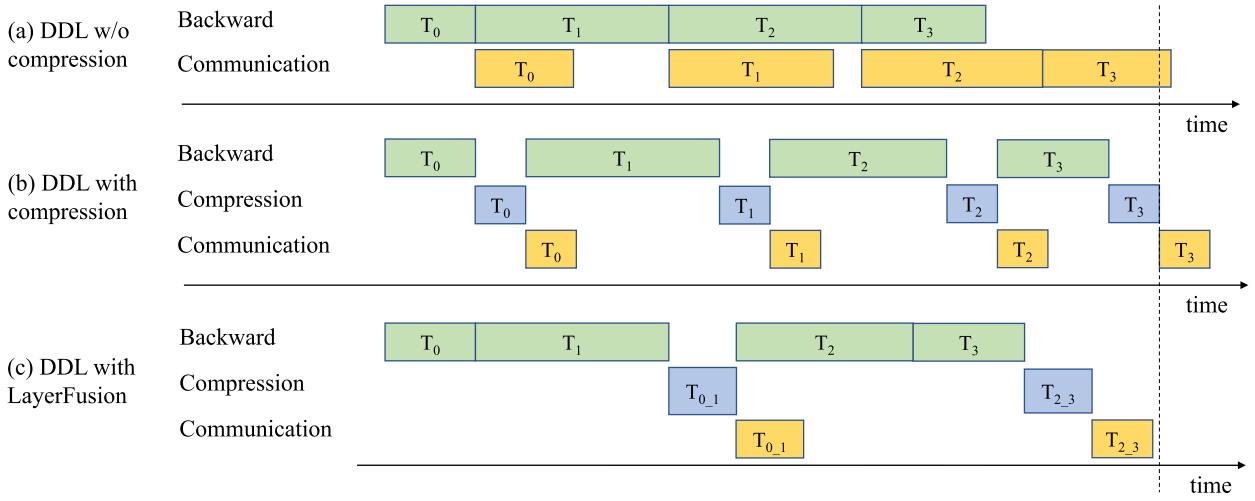


Fig. 2. Distributed training with different methods.

Table 1
The time for compression and decompression of gradient compression methods for ResNet-50 on V100 GPUs.

Type	Method	$T_{comp\_decomp}(ms)$
Sparsification	Top-K - 1%	122
	DGC - 1%	242
	Random-K - 1%	187
Quantization	QSGD - 2 bit	46
	SignSGD	13
	TernGrad	70
Low Rank	PowerSGD - Rank 4	174

tensors to synchronize, as illustrated in Fig. 1b. Given these observations, we believe that the compression of the gradient of the DNN model layer by layer can lead to significant compressing overhead, which is a significant reason for the poor performance of the compression algorithm.

The gradient compression algorithm necessitates a trade-off between model accuracy and compression rate. When the compression rate of the algorithm is excessively high, it can improve the training speed, but the loss of model accuracy will be substantial. Conversely, if the compression rate is too low, the model's accuracy is ensured, but the training speed is not improved. Therefore, selecting an imbalanced hyperparameter, like the parameter k in Top-k, results in poor performance.

The use of layer-wise gradient compression algorithms can lead to significant compression and communication overhead, which can negatively impact the efficiency of the distributed training process. The communication process incurs a fixed overhead during startup, as documented in [41], and there is also a fixed overhead for starting and executing the kernel in CUDA [5]. However, our experiments have shown that most gradient compression algorithms are not significantly affected by the size of the gradient tensor, with the compression overhead not doubling even as the tensor size increases from 2^4 to 2^{20}. This observation has led us to develop a strategy for optimizing the layer-wise method by fusing multiple gradient tensors, which can substantially reduce the compression overhead and improve the efficiency of the distributed training process.

As shown in Fig. 2, we need to find a gradient fusion strategy to make the gradient compression algorithm optimal. However, overlap between the computation and communication needs to be considered when fusing multiple gradient tensors. For example, an extreme case would be to fuse the gradient of the entire model into a single tensor. Although the compression algorithm will only be called once, the communication process cannot be started before the computation is completed, and there is no overlap between the calculation and communication process, resulting in extremely low training efficiency.

To achieve optimal results, the design of a gradient fusion strategy should consider various factors, including the compression algorithms utilized, the characteristics of the deep neural network models, and the cluster configuration, such as the network bandwidth and the num-

ber of workers. To enhance the actual performance of the compression algorithm, we propose the LayerFusion algorithm. Additionally, to enhance the training efficiency and scalability of DDL systems, we propose the $Sketch-Fusion$ SGD. These methods aim to improve the overall training efficiency by reducing the compression and communication overhead while maintaining a high level of accuracy.

4. Method

In this section, we introduce our method, which aims to improve the scalability and model convergence efficiency of distributed training. Firstly, we propose the $Sketch-Fusion$ SGD, which is designed to enhance the scalability and training speed of distributed training. We then present LayerFusion, which is employed in $Sketch-Fusion$ SGD to optimize the scalability and convergence efficiency of gradient compression algorithms. Finally, we demonstrate how LayerFusion can automatically determine an optimal gradient fusion strategy without introducing additional hyperparameters. Additionally, we provide theoretical proof of the convergence rate of our proposed method.

4.1. Overview

In $Sketch-Fusion$ SGD, each server transmits the sketch of the gradient tensor instead of the entire or compressed gradient tensor. To achieve this, each server compresses its gradient into a $Count-Sketch$ structure locally. During the aggregation phase, the servers add up their compressed gradient sketches. Then, in Line 7, the algorithm recovers the top-k largest gradient elements by magnitude from the summed sketch and uses the exact values of these top-k elements to update the weight. The algorithm used to recover the top-k elements from the summed sketch is called HEAVYMIX, which was introduced in [17]. This approach can improve the scalability and convergence efficiency of distributed training.

Algorithm 1 Sketch-Fusion SGD.

Input: dataset D; model $\boldsymbol{X} = \{x_1, ..., x_y\}$; learning rate η; the number of iterations to train: T; the sparsity rate: k; the number of workers: N;

1: **for** $t = 0 \rightarrow T$ **do**
2: Sampling a mini-batch of data D_t from D;
3: **for** $i = 0 \rightarrow y$ **do**
4: Compute stochastic gradient $\widetilde{G}_{i,t}$;
5: Compute sketches S_t^g of $\widetilde{G}_{i,t}$;
6: Aggregate sketches $S_t = \frac{1}{N}\sum_{p=1}^{N} S_t^p$;
7: Compute decompressed gradient $G_{i,t} = HEAVYMIX(S_t, k)$; [17]
8: $x_{i,t+1} = x_{i,t} - \eta G_{i,t}$
9: **end for**
10: **end for**

Effectively reduces the computational overhead. Moreover, communication and computation are overlapped to further reduce the communication overhead.

In LayerFusion, the key idea is to reduce the number of calls to the gradient compression algorithm. This is achieved by fusing the gradient tensors, which reduces computational overhead. Moreover, communication and computation are overlapped to further reduce the communication overhead.

Specifically, LayerFusion divides a model X into y groups, i.e., $\boldsymbol{X} = x_1, ..., x_y$, and gradient tensors in the same group are fused and compressed together in a compression operation. T is the total number of training iterations, and $x_{i,t}$ represents x_i in the t_{th} iteration. The gradient compression algorithm $C(\cdot)$ may include methods such as sparsification, quantization, or low-rank approximation, and $C^{-1}(\cdot)$ is the corresponding decompression algorithm.

Moreover, LayerFusion supports a series of communication schemes for different compression algorithms, including all-reduce [39,34], allgather [46], and parameter server [25]. Communication is overlapped with backpropagation to minimize the communication overhead. After communication, the fused and compressed gradients are decompressed and aggregated to update the model. The entire process is summarized in Algorithm 2.

Algorithm 2 LayerFusion for compression methods.

Input: dataset D; model $\boldsymbol{X} = \{x_1, ..., x_y\}$; learning rate η; compression function $C(\cdot)$; the number of iterations to train: T

1: **for** $t = 0 \rightarrow T$ **do**
2: Sampling a mini-batch of data D_t from D;
3: **for** $i = 0 \rightarrow y$ **do**
4: Compute stochastic gradient $\widetilde{G}_{i,t}$;
5: Compute compressed gradient $\delta_{i,t} = C(\widetilde{G}_{i,t})$;
6: Communicate compressed gradients $\Delta_{i,t} = communicate(\delta_{i,t})$;
7: Aggregate gradients from all workers $G_{i,t} = aggregate(C^{-1}(\Delta_{i,t}))$;
8: $x_{i,t+1} = x_{i,t} - \eta G_{i,t}$
9: **end for**
10: **end for**

4.2. Theoretical guarantee

In this section, we provide an analysis of the convergence rate of the $Sketch-Fusion$ SGD method with the sparsified gradient compression algorithm using LayerFusion. We should note that the analysis presented below is specifically focused on synchronous data-parallel distributed SGD.

4.2.1. Preliminaries

We define $F(\cdot)$ as the loss function we need to optimize, and the stochastic optimization problem we study can be expressed as:

$$\min f(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi \sim D_i} F_i(x;\xi) \tag{2}$$

Here, n is the number of workers in the cluster, D_i is a subset of the training dataset on worker i, and $F_i(x;\xi)$ is the loss computed from data on worker i.

In this subsection, we use assumptions for the loss function and the variance of the stochastic gradient to analyze the convergence rate of the $Sketch-Fusion$ SGD method. The following symbols are used: $|\cdot|_2$ denotes the L2 norm of a tensor or the spectral norm of a matrix; $\bigtriangledown f(\cdot)$ denotes the gradient of a function f; 1_n denotes the column tensor in $\mathbb{R}^n$ with 1 for all elements; and f^* denotes the optimal solution for the stochastic optimization problem.

Assumption 1 *(Lipschiz continuity).* $\bigtriangledown f(\cdot)'$ is Lipschitz continuous with respect to the L2 norm, i.e.,

$$\left\| \bigtriangledown f_i(x) - \bigtriangledown f_i(y) \right\|_2 \leq L \left\| x - y \right\|_2 \qquad \forall x, \forall y, \forall i. \tag{3}$$

Assumption 2 *(Bounded variance).* The variance of stochastic gradient is bounded, i.e.,

$$\mathbb{E}_{\xi \sim D_i} \left\| \bigtriangledown F_i(x;\xi) - \bigtriangledown f_i(x) \right\|_2^2 \leq \sigma^2 \qquad \forall x, \forall i. \tag{4}$$

Assumption 3 *(Unbiasedness).* The stochastic gradient of $F_i(x;\xi)$ is unbiased, i.e.,

$$\mathbb{E}_{\xi \sim D_i} \bigtriangledown F_i(x;\xi) = \bigtriangledown f_i(x) \qquad \forall x, \forall i. \tag{5}$$

Assumption 4. Gradient sparsification algorithms can exchange all the gradients in any p consecutive iterations.

These assumptions are commonly employed in previous works to analyze the convergence rate of distributed SGD [19,3,11,17,29].

Table 2
Frequantly used notations.

Name	Description
N	the number of tensors of a DNN model
β	the computation time of each iteration
y	the number of groups into which the model is divided
x_i	the size of the i_{th} group
X_y	a multi-layer fusion strategy of the DNN model
$Comp(x_i)$	the compression time of the i_{th} group
$Comm(x_i)$	the communication time of the i_{th} group
$p(x_i)$	the overlap time between the computation time and communication time

4.2.2. Sparsified communication

Partial gradient transmission achieves the same convergence rate as vanilla distributed SGD. It has been shown in previous works on gradient sparsification [44,17,20,6]. Since LayerFusion selects an essential partial gradient tensor for communication, it can maintain the same convergence rate as the applied gradient sparsification algorithms.

Theorem 1. *In data-parallel distributed training, all workers share the same training dataset. When $\gamma = \alpha\sqrt{B/T}$, where $\alpha > 0$ is a constant, B is the total mini-batch size on all workers, $L_\gamma \leq 1$, and $6np^2L^2\gamma^2 < 1$, LayerFusion for gradient sparsification algorithms has the convergence rate as shown in the following equation:*

$$\frac{1}{T}\left(\sum_{t=0}^{T-1}\left|\nabla f\left(\frac{X_t 1_n}{n}\right)\right|_2^2\right) \leq \frac{4\alpha^{-1}(f(x_0)-f^*)+2\alpha L\sigma^2}{\sqrt{BT}} + \frac{2pn^2\alpha^2L^2\sigma^2}{T}, \tag{6}$$

when the number of iterations satisfies $T \geq 12nB\alpha^2p^2L^2$. Previous work [19] has proven Theorem 1 under a different assumption of the training dataset. Here, $\frac{X_t 1_n}{n}$ is the average of the gradients on all workers. If the number of iterations T is large enough, the right side of Equation (6) is dominated by its first term. And the LayerFusion algorithm for gradient sparsification achieves the same $O(\frac{1}{\sqrt{BT}})$ convergence rate as vanilla distributed SGD [44]. Please refer to the appendix for the proof of quantized communication.

4.3. Method for searching the optimal fusion strategy

To determine the optimal computation-communication overlap strategy, previous work has abstracted the problem as an optimization problem [41]. Similarly, we have abstracted the fusion problem as an optimization problem and propose our algorithm, LayerFusion, for searching the optimal gradient fusion strategy. Table 2 lists the common notations we use.

The goal of LayerFusion is to minimize the iteration time of compression algorithms. Using the WFBP method to overlap computation and communication can reduce the overall iteration time. Therefore, the target formula for this problem is:

$$\min_{y\in[1,N]}\min_{X_y\in\mathcal{X}_y} F(X_y) = \sum_{i=1}^{y} Comp(x_i) + \sum_{i=1}^{y} Comm(x_i) + \beta - \sum_{i=1}^{y} p(x_i) \tag{7}$$

Here, $\mathcal{X}_y$ is the set of all possible fusion strategies with y groups. LayerFusion heuristically searches for an optimal multi-layer fusion strategy, but we have found that the gain of LayerFusion decreases as the number of groups of gradient fusion increases for most models and compression algorithms. To address this, we introduce prior knowledge into the heuristic search algorithm. If the performance of strategy $F(X_y)$ is worse than $F(X_{y-1})$ or the difference is less than $\epsilon F_{min}(X_{y-1})$ (where ϵ is a constant parameter), the algorithm terminates, and X_y is returned. Moreover, LayerFusion adopts binary search when searching X_y, so the time complexity is $O(N \log N)$.

Table 3
The experimental setup of training deep models.

Model	Dataset	Epochs	b	η
ResNet-20	CIFAR10	140	128	0.1
VGG-16	CIFAR10	140	128	0.1
ResNet-50	ImageNet	140	128	0.01
LSTM	PTB	65	32	0.01

5. Experiment

To thoroughly evaluate the effectiveness and generalizability of our proposed method, we conducted experiments using various representative DNNs from different AI domains and with different datasets. Specifically, we chose the CIFAR10 dataset [23] consisting of 50,000 training samples, the ImageNet dataset [37] containing about 1.2 million images for classification, and the Penn TreeBank (PTB) dataset [32] in the natural language processing field, containing 923,000 language samples. For each dataset, we selected a neural network model with high accuracy and computational efficiency. For CIFAR10, we used the ResNet-20 [15] and VGG-16 [43] models, and for ImageNet, we employed the ResNet-50 [15] model. The PTB dataset was tested on an LSTM language model with two hidden layers [42]. We chose these DNNs because they were used in the original papers of the compared methods for comparative experiments [33,30,48,3]. The training details and hyperparameters for the models are listed in Table 3, where b denotes the batch size, and η refers to the learning rate. All models were trained with 32-bit floating-point precision.

We conducted all experiments on a cluster of 16 NVIDIA Tesla V100 GPUs with 32 GB memory and two 24-core/48-thread Intel Xeon Gold 6252 2.1 GHz processors. The servers in the cluster are running the Ubuntu 18.04 LTS system and are equipped with PyTorch-1.8.1, Horovod-0.22.1, CUDA-11.1, OpenMPI-4.1.3, and NCCL-2.8.3, all connected by Ethernet with 20 Gbps bandwidth.

5.1. Training speed improvement

We conducted an experiment using LayerFusion on eight gradient compression algorithms and our proposed $Sketch-Fusion$ SGD on three different DNN models using different datasets. Figs. 3-5 show the performance of various gradient compression algorithms with LayerFusion. The light-colored cylinder in the figure represents the improvement of the gradient compression algorithm with LayerFusion. Among them, the Top-k algorithm improved by 1.58×-2.51× in the task scenario of ResNet-20 on CIFAR10. Additionally, almost all gradient compression algorithms using LayerFusion exceeded the baseline performance. The $Sketch-Fusion$ SGD achieved the optimal scaling factor in different task scenarios and different numbers of cluster GPUs. Unfortunately, there was no significant improvement in the PowerSGD algorithm. We believe the main issue lies in the calculation of the low-rank matrix, which remains a bottleneck.

LayerFusion applied to popular FP16 compression (semi-precision training) method achieved about 50% scaling factor on ResNet-50 on the ImageNet image classification task with 16 GPUs. This is a significant increase from the baseline (FP32) scaling factor of only about 35%. Surprisingly, LayerFusion also significantly improved the 1-bit quantization methods. The scaling factor of SignSGD and EFSignSGD for LSTM was up to 1.41×-2.87× higher than that of the baseline and layer-wise compression. The corresponding improvements for ResNet-20 and ResNet-50 were up to 1.40×-2.47× and 1.34×-2.36×.

We found that the gradient compression algorithms provided a lower scaling factor improvement for ResNet-20 and LSTM compared to the scaling factor in ResNet-50. We believe this is due to the small number of model layers of ResNet-20 and LSTM, and layer-wise compression overhead is not a bottleneck for scalability.

To evaluate the improvement of training speed brought by the gradient compression algorithms, we counted the throughput of training

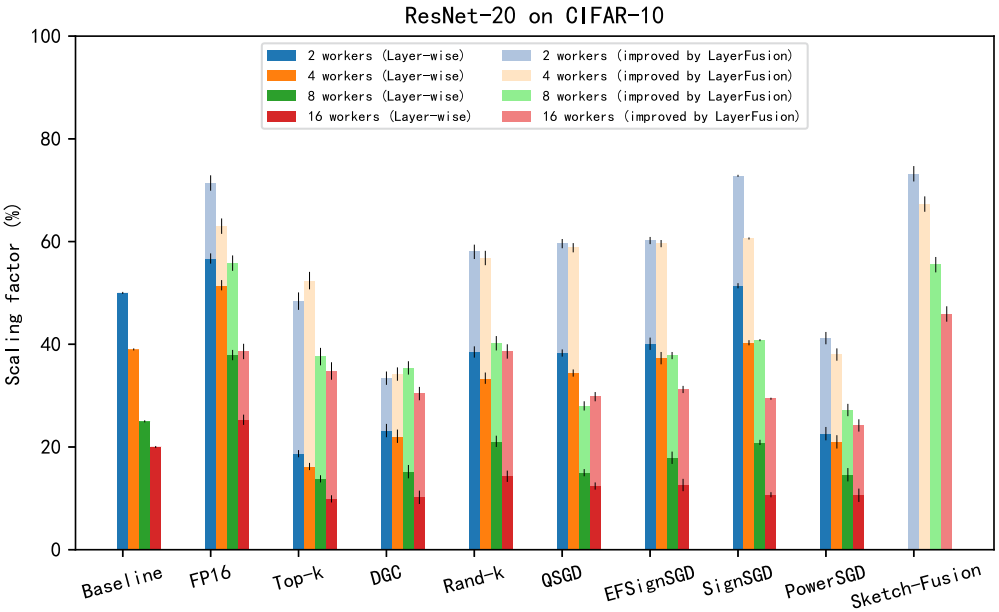


Fig. 3. The performance of ResNet-20 on CIFAR10.

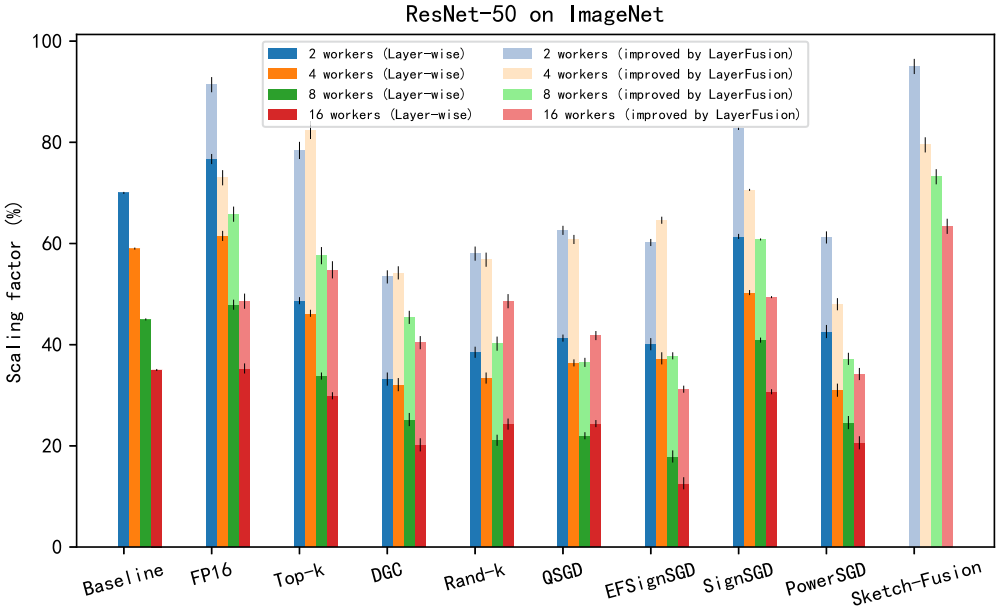


Fig. 4. The performance of ResNet-50 on ImageNet.

using different algorithms, which is shown in Table 4. It can be seen that the throughput of $Sketch-Fusion$ SGD on the three models is significantly ahead of other gradient compression algorithms. Furthermore, the throughput of all eight gradient compression algorithms, has been improved to some extent after the application of LayerFusion. Among them, DGC achieved the most significant improvement, reaching 38% (for ResNet-20 on CIFAR10).

5.2. Convergence efficiency

We conducted a comparative experiment to evaluate the convergence efficiency of the gradient compression algorithm. Specifically, we trained ResNet-20 on CIFAR10 and ResNet-50 on ImageNet using our $Sketch-Fusion$ SGD and DGC [30], the state-of-the-art algorithm for training efficiency. The time-wise top-1 accuracy curves obtained during the training process are depicted in Fig. 6. The results indicate that the time required for convergence with $Sketch-Fusion$ SGD was reduced by 5.2-18.6% compared to the baseline (FP32, full-precision training) and more than 20% compared to DGC. The experiment results showed that ResNet-50 on ImageNet with DGC failed to converge under our experimental settings.

6. Conclusion

In this paper, we propose the $Sketch-Fusion$ SGD method, based on the Count-Sketch data structure, to further improve the scalability and training efficiency of DDL systems. We prove the convergence rate of $Sketch-Fusion$ SGD and demonstrate through extensive experiments that it has the best scalability and improves training efficiency by up to 18.6% compared to other gradient compression algorithms. We also introduce LayerFusion, a gradient fusion algorithm in $Sketch-Fusion$ SGD that accelerates the distributed training of deep

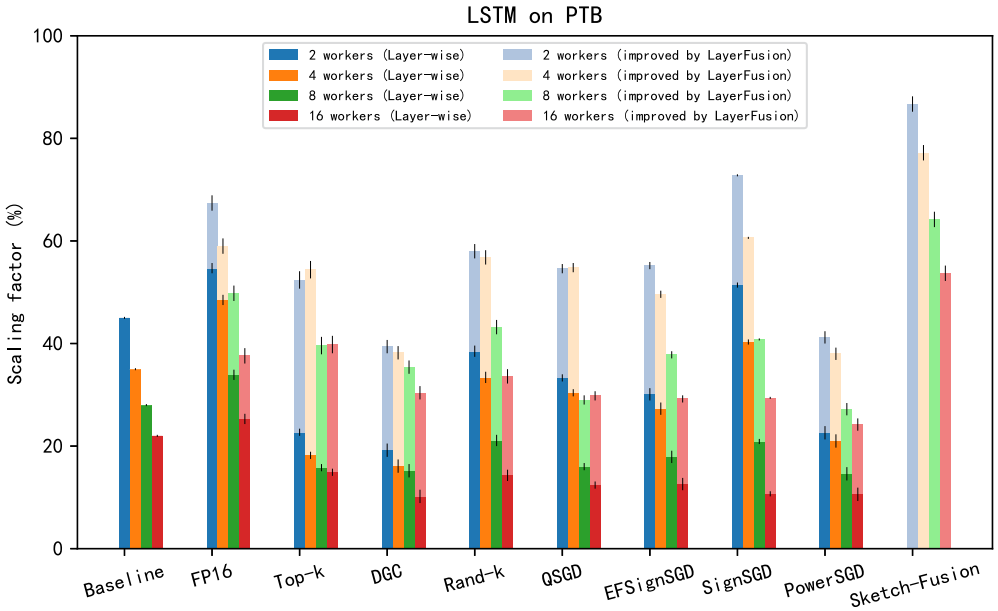


Fig. 5. The performance of LSTM on PTB.

Table 4
The system training throughput on a 16-GPU cluster.

Method	Layer-wise			LayerFusion		
	ResNet-20	ResNet-50	LSTM	ResNet-20	ResNet-50	LSTM
Top-k	976	428	1004	1073(+97)	530(+102)	1255(+251)
DGC	3416	1768	3572	4716(+1300)	2174(+406)	4136(+564)
Rand-k	1040	522	1227	1317(+277)	600(+78)	1476(+249)
QSGD	4011	2319	3471	5213(+1202)	3001(+682)	4484(+1013)
EFSignSGD	3294	2038	2719	3418(+124)	2417(+379)	3042(+323)
SignSGD	5240	3321	5864	6133(+893)	4187(+866)	6742(+878)
PowerSGD	1145	537	957	1371(+226)	542(+5)	1085(+128)
Sketch-Fusion SGD	-	-	-	**8430**	**6321**	**9864**

Note: The throughput is measured with processed images per second.

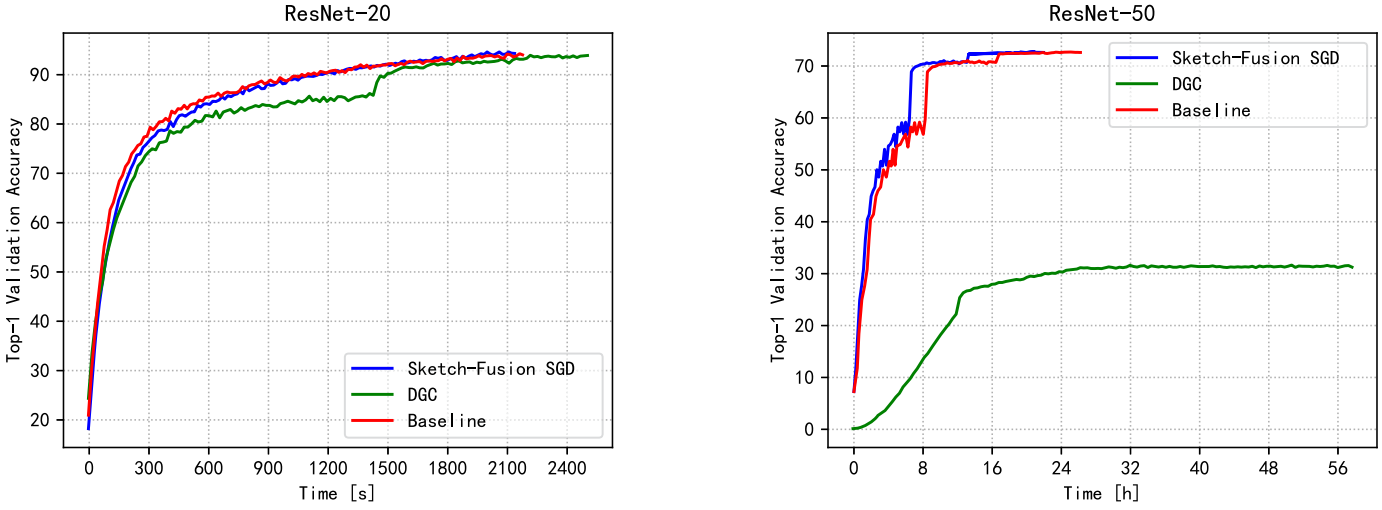


Fig. 6. Validation top-1 accuracy curves of ResNet-20 and ResNet-50 trained on CIFAR10 dataset and ImageNet dataset with $Sketch-Fusion$ SGD, DGC (Layer-wise) and Baseline (FP32).

neural networks and improves the performance of gradient compression algorithms. By using a heuristic search algorithm, LayerFusion can develop an optimal gradient grouping strategy without introducing extra hyperparameters, thus optimizing the performance of gradient compression algorithms. Our experiments show that LayerFusion can greatly improve the scalability of different gradient compression algorithms without sacrificing the accuracy of DNN models. In future work, considering the role of CPU in being able to offload part of the computation process in distributed training scenarios may significantly improve the effectiveness of the $Sketch-Fusion$ SGD method.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Appendix A

Assumption 5. The variance of gradient among workers is bounded, that is

$$\mathbb{E}_{\xi\sim\mu_{i,n}}\left\|\nabla f_i(x)-\nabla f_{(x)}\right\|_2^2\le\zeta^2\qquad\forall x, \tag{A.1}$$

where $\mu(1,n)$ is a discrete uniform distribution of integers from 1 to n. If all workers share the same training data, $\zeta=0$.

A.1. Proof to Theorem 1

We refer to a corollary [19] and restate it here as:

Corollary 1 *([19]). Under Assumptions 1-5, if we set $\gamma=\alpha\sqrt{B/T}$ where $\alpha>0$ is a constant, then the convergence rate for sparsification algorithms is given by:*

$$\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(\frac{X_t 1_n}{n})\right\|_2^2)\le\frac{4\alpha^{-1}(f(x_0)-f^*)+2\alpha L\sigma^2}{\sqrt{BT}}+\frac{2pn^2\alpha^2L^2\sigma^2+6nB\alpha^2p^2L^2\zeta^2}{T}, \tag{A.2}$$

if the number of iterations satisfies $T\ge 12nB\alpha^2p^2L^2$.

Since we assume that all workers share the same training dataset, we can remove the items with ζ in Inequality (A.2) and obtain Theorem 1.

A.2. Proof to quantize communication

$Q(\cdot)$ is defined as the quantization compressor. The bound of the expected error of $Q(\cdot)$ is defined as $q_i=\sup\frac{\|Q(x_i)-x_i\|_2^2}{\|x_i\|_2^2}$, where x_i represents the gradients in the i_th group. We also define $q=\max\{q_i\}$ to derive the convergence rate of quantization algorithms with LayerFusion under Assumption 1-3.

Theorem 2. *If all workers share the same training dataset and error feedback is applied, and we set $\gamma=\alpha\sqrt{B/T}$ where $\alpha>0$ is a constant and $(1+\frac{q}{n})L_\gamma<2$, LayerFusion for quantization algorithms has the convergence rate given by:*

$$\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_t)\right\|_2^2)\le\frac{2\alpha^{-1}(f(x_0)-f^*)+(1+q)\alpha L\sigma^2 y}{\sqrt{BT}}, \tag{A.3}$$

if the number of iterations T satisfies $T\ge B\alpha^2L^2(1+\frac{q}{n})^2$. Thus, LayerFusion for quantization algorithm can achieve the convergence rate of $O(1/\sqrt{BT})$.

The proof relies on the following corollary [19], which we restate:

Corollary 2 *([19]). Under Assumption 1-5, if a quantization function with an error bound of q is used and $\gamma=\alpha\sqrt{B/T}$ where $\alpha>0$ is a constant, then the following convergence rate holds for quantization algorithms:*

$$\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_t)\right\|_2^2)\le\frac{2\alpha^{-1}(f(x_0)-f^*)+(1+q)\alpha L\sigma^2}{\sqrt{BT}}+\frac{b}{\sqrt{BT}}\alpha qL\zeta^2, \tag{A.4}$$

if the number of iterations T satisfies $T\ge B\alpha^2L^2(1+\frac{q}{n})^2$. Since we assume all workers share the same training dataset, the term with ζ can be removed from Inequality (A.4), giving:

$$\begin{aligned}\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_{i,t})\right\|_2^2)&\le\frac{2\alpha^{-1}(f(x_0)-f^*)+(1+q_i)\alpha L\sigma^2}{\sqrt{BT}}+\frac{b}{\sqrt{BT}}\alpha qL\zeta^2\\&\le\frac{2\alpha^{-1}(f(x_0)-f^*)+(1+q)\alpha L\sigma^2}{\sqrt{BT}}+\frac{b}{\sqrt{BT}}\alpha qL\zeta^2,\end{aligned} \tag{A.5}$$

Here, $x_{i,t}$ represents the value of x_i at the t_{th} iteration, and $q=\max q_i$. Since $|\nabla f(x)|_2^2=\sum_{i=1}^y|\nabla f(x_i)|_2^2$, we can simplify further and obtain:

$$\begin{aligned}\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_t)\right\|_2^2)&=\sum_{i=1}^{y}\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_{i,t})\right\|_2^2)\\&\le\frac{2\alpha^{-1}(\sum_{i=1}^y f(x_{i,0})-\sum_{i=1}^y f^*(x_i))+(1+q)\alpha L\sigma^2 y}{\sqrt{BT}}\\&\le\frac{2\alpha^{-1}(f(x_0)-\sum_{i=1}^y f^*)+(1+q)\alpha L\sigma^2 y}{\sqrt{BT}}.\end{aligned} \tag{A.6}$$

Since F^* is the optimal solution, we have $f^{\le}\sum_{i=1}^y f^*(x_i)$. We can rewrite Inequality (A.6) as follows:

$$\frac{1}{T}(\sum_{t=0}^{T-1}\left\|\nabla f(x_t)\right\|_2^2)\le\frac{2\alpha^{-1}(f(x_0)-f^*)+(1+q)\alpha L\sigma^2 y}{\sqrt{BT}}. \tag{A.7}$$

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv:1603.04467, 2016.

[2] A.F. Aji, K. Heafield, Sparse communication for distributed gradient descent, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 440–445.

[3] D. Alistarh, D. Grubic, J. Li, R. Tomioka, M. Vojnovic, QSGD: communication-efficient SGD via gradient quantization and encoding, Adv. Neural Inf. Process. Syst. 30 (2017) 1709–1720.

[4] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, C. Renggli, The convergence of sparsified gradient methods, Adv. Neural Inf. Process. Syst. 31 (2018) 5973–5983.

[5] Y. Arafa, A.H.A. Badawy, G. Chennupati, N. Santhi, S. Eidenbenz, Low overhead instruction latency characterization for nvidia gpgpus, in: 2019 IEEE High Performance Extreme Computing Conference, 2019, pp. 1–8.

[6] D. Basu, D. Data, C. Karakus, S. Diggavi, Qsparse-local-SGD: distributed SGD with quantization, sparsification and local computations, Adv. Neural Inf. Process. Syst. 32 (2019).

[7] J. Bernstein, Y.X. Wang, K. Azizzadenesheli, A. Anandkumar, signSGD: compressed optimisation for non-convex problems, in: International Conference on Machine Learning, 2018, pp. 560–569.

[8] J. Chen, X. Pan, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous SGD, arXiv:1604.00981, 2017.

[9] J. Choquette, W. Gandhi, NVIDIA A100 GPU: performance & innovation for GPU computing, in: 2020 IEEE Hot Chips 32 Symposium, 2020, pp. 1–43.

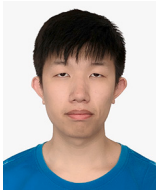
[10] T. Dettmers, 8-bit approximations for parallelism in deep learning, arXiv:1511.04561, 2016.

[11] S. Ghadimi, G. Lan, Stochastic first- and zeroth-order methods for nonconvex stochastic programming, SIAM J. Optim. 23 (2013) 2341–2368.

[12] P. Goyal, P. Dollár, R.B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: training imagenet in 1 hour, arXiv:1706.02677, 2018.

[13] M. Gupta, V. Varma, S. Damani, K.N. Narahari, Compression of deep learning models for NLP, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 3507–3508.

[14] V. Gupta, D. Choudhary, P.T.P. Tang, X. Wei, X. Wang, Y. Huang, A. Kejariwal, K. Ramchandran, M.W. Mahoney, Fast distributed training of deep neural networks: dynamic communication thresholding for model and data parallelism, arXiv:2010.08899.
[15] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (2017) 6869–6898.
[17] N. Ivkin, D. Rothchild, E. Ullah, I. Stoica, R. Arora, et al., Communication-efficient distributed SGD with sketching, Adv. Neural Inf. Process. Syst. 32 (2019) 13144–13154.
[18] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, X. Chu, Highly scalable deep learning training system with mixed-precision: training imagenet in four minutes, arXiv:1807.11205, 2018.
[19] P. Jiang, G. Agrawal, A linear speedup analysis of distributed deep learning with sparse and quantized communication, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 2530–2541.
[20] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, C. Guo, A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters, in: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, 2020, pp. 463–479.
[21] N. Jouppi, C. Young, et al., In-datacenter performance analysis of a tensor processing unit, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.
[22] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A.C. Berg, W.Y. Lo, P. Dollár, R. Girshick, Segment anything, arXiv:2304.02643, 2023.
[23] A. Krizhevsky, V. Nair, G. Hinton, Cifar-10, Canadian Institute for Advanced Research, 2010, 5, 4, http://www.cs.toronto.edu/kriz/cifar.html.
[24] Y. LeCun, Y. Bengio, G.E. Hinton, Deep learning, Nature 521 (2015) 436–444.
[25] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.Y. Su, Scaling distributed machine learning with the parameter server, in: 11th {USENIX} Symposium on Operating Systems Design and Implementation, 2014, pp. 583–598.
[26] M. Li, D.G. Andersen, A.J. Smola, K. Yu, Communication efficient distributed machine learning with the parameter server, in: Proceedings of the 27th International Conference on Neural Information Processing Systems, 2014, pp. 19–27.
[27] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, S. Chintala, Pytorch distributed: experiences on accelerating data parallel training, arXiv:2006.15704, 2020.
[28] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, N.S. Kim, A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture, 2018, pp. 175–188.
[29] X. Lian, C. Zhang, H. Zhang, C.J. Hsieh, W. Zhang, J. Liu, Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent, Adv. Neural Inf. Process. Syst. 30 (2017).
[30] Y. Lin, S. Han, H. Mao, Y. Wang, W.J. Dally, Deep gradient compression: reducing the communication bandwidth for distributed training, arXiv:1712.01887, 2020.
[31] A.I. Maqueda, A. Loquercio, G. Gallego, N. García, D. Scaramuzza, Event-based vision meets deep learning on steering prediction for self-driving cars, in: 2018 IEEECVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 5419–5427.
[32] M. Marcus, B. Santorini, M.A. Marcinkiewicz, Building a large annotated corpus of English: the Penn Treebank, Technical Report, 1993.
[33] K. Mishchenko, B. Wang, D. Kovalev, P. Richtárik, IntSGD: adaptive floatless compression of stochastic gradients, arXiv:2102.08374, 2022.
[34] P. Patarasuk, X. Yuan, Bandwidth optimal all-reduce algorithms for clusters of workstations, J. Parallel Distrib. Comput. 69 (2009) 117–124.
[35] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, C. Guo, A generic communication scheduler for distributed DNN training acceleration, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 16–29.
[36] J. Redmon, S. Divvala, R.B. Girshick, A. Farhadi, You only look once: unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788.
[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M.S. Bernstein, A. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (2015) 211–252.
[38] A. Sapio, M. Canini, C.Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D.R.K. Ports, P. Richtárik, Scaling distributed machine learning with in-network aggregation, arXiv:1903.06701, 2020.
[39] A. Sergeev, M.D. Balso, Horovod: fast and easy distributed deep learning in tensorflow, arXiv:1802.05799, 2018.
[40] S. Shi, X. Chu, B. Li, Exploiting simultaneous communications to accelerate data parallel distributed deep learning, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, 2021, pp. 1–10.
[41] S. Shi, Q. Wang, X. Chu, B. Li, Y. Qin, R. Liu, X. Zhao, Communication-efficient distributed deep learning with merged gradient sparsification on GPUs, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, 2020, pp. 406–415.
[42] X. Shi, Z. Chen, H. Wang, D.Y. Yeung, W.k Wong, W.c Woo, Convolutional lstm network: a machine learning approach for precipitation nowcasting, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 1, 2015, pp. 802–810.
[43] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556, 2015.
[44] S.U. Stich, J.B. Cordonnier, M. Jaggi, Sparsified SGD with memory, Adv. Neural Inf. Process. Syst. 31 (2018) 4452–4463.
[45] N. Strom, Scalable distributed DNN training using commodity GPU cloud computing, in: Interspeech 2015, 2015, pp. 1488–1492.
[46] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of collective communication operations in mpich, Int. J. High Perform. Comput. Appl. 19 (2005) 49–66.
[47] Y. Tsuzuku, H. Imachi, T. Akiba, Variance-based gradient compression for efficient distributed deep learning, arXiv:1802.06058, 2018.
[48] T. Vogels, S.P.R. Karimireddy, M. Jaggi, PowerSGD: practical low-rank gradient compression for distributed optimization, Adv. Neural Inf. Process. Syst. 32 (2019) 14236–14245.
[49] W. Wang, N. Srebro, Stochastic nonconvex optimization with large minibatches, in: Algorithmic Learning Theory, 2019, pp. 857–882.
[50] J. Wu, W. Huang, J. Huang, T. Zhang, Error compensated quantized SGD and its applications to large-scale distributed optimization, in: International Conference on Machine Learning, 2018, pp. 5325–5333.
[51] E. Xu, Huawei launches ascend 910, the world's most powerful ai processor, and mindspore, an all-scenario ai computing framework, https://www.huawei.com/en/news/2019/8/huawei-ascend-910-most-powerful-ai-processor. (Accessed 19 July 2021), 2019, Online.
[52] H. Xu, C.Y. Ho, A.M. Abdelmoniem, A. Dutta, E.H. Bergou, K. Karatsenidis, M. Canini, P. Kalnis, Compressed communication for distributed deep learning: Survey and quantitative evaluation, Technical Report, 2020.
[53] Y. Xu, X. Liu, X. Cao, C. Huang, E. Liu, S. Qian, X. Liu, Y. Wu, F. Dong, C.W. Qiu, et al., Artificial intelligence: a powerful paradigm for scientific research, Innovation 2 (2021) 100179.
[54] Y. You, Z. Zhang, C.J. Hsieh, J. Demmel, K. Keutzer, Imagenet training in minutes, in: Proceedings of the 47th International Conference on Parallel Processing, 2018, pp. 1–10.
[55] L. Zhang, S. Shi, X. Chu, W. Wang, B. Li, C. Liu, Dear: accelerating distributed deep learning with fine-grained all-reduce pipelining, arXiv:2302.12445, 2023.
[56] Z. Zhang, C. Chang, H. Lin, Y. Wang, R. Arora, X. Jin, Is network the bottleneck of distributed training?, in: Proceedings of the Workshop on Network Meets AI & ML, 2020, pp. 8–13.
[57] Z. Zong, G. Song, Y. Liu, Detrs with collaborative hybrid assignments training, arXiv:2211.12860, 2023.

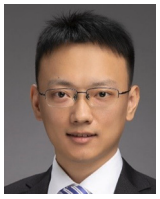


Lingfei Dai received his B.Eng. degree from China University of Geosciences (Wuhan) in 2020. He is currently pursuing a master's degree at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include edge intelligence and distributed training.

Luqi Gong received his B.Eng. degree from Shandong University of Science and Technology in 2017 and the M.Eng. degree from the Chinese Academy of Sciences in 2020. He works as an engineer from 2020 and his research interests include deep neural network acceleration and compression.

Zhulin An received the B.Eng. and M.Eng. degrees in computer science from Hefei University of Technology, Hefei, China, in 2003 and 2006, respectively and the Ph.D. degree from the Chinese Academy of Sciences, Beijing, China, in 2010. He is currently an associate professor of Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include optimization of deep neural network and lifelong learning.

Yongjun Xu received his B.Eng. and Ph.D. degree in computer communication from Xi'an Institute of Posts & Telecoms (China) in 2001 and Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China in 2006, respectively. He is a professor at Institute of Computing Technology, Chinese Academy of Sciences in Beijing, China. His current research interests include artificial intelligence systems, and big data processing.



Boyu Diao received the B.Eng. degree in Computer Science from Beijing Institute of Technology, Beijing, China in 2012 and the Ph.D. degree in Computer Architecture from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He is currently an associate professor at Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include efficient and reliable machine learning systems, and edge intelligence.