Efficient Distributed Stochastic Gradient Descent Through Gaussian Averaging

Kaifan Hu(✉), Chengkun Wu, and En Zhu

School of Computer, National University of Defense Technology, Changsha 410073, Hunan, China
{kaifan_hu,chengkun_wu,enzhu}@nudt.edu.cn

Abstract. Training of large-scale machine learning models presents a hefty communication challenge to the Stochastic Gradient Descent (SGD) algorithm. In a distributed computing environment, frequent exchanges of gradient parameters between computational nodes are inevitable in model training, which introduces enormous communication overhead. To improve communication efficiency, a gradient compression technique represented by gradient sparseness and gradient quantization is proposed. Base on that, we proposed a novel approach named *Gaussian Averaging SGD* (GASGD), which transmits 32 bits between nodes in one iteration and achieves a communication complexity of $\mathcal{O}(1)$. A theoretical analysis demonstrates that GASGD has a similar convergence performance compared with other distributed algorithms with a significantly smaller communication cost. Our experiments validate the theoretical conclusion and demonstrate that GASGD significantly reduces the communication traffic per worker.

Keywords: Communication-efficient · Distributed optimization · Parallel computing

1 Introduction

Deep learning is widely used in various fields. Recently, more complex and lager models with over millions and even billions of parameters have appeared. i.e., BERT [7] (1.1B). A simple way to train such a model effectively is to use a distributed SGD algorithm. It can be formulated as below, where w_t represents the parameters of the model in iteration t, η_t is the learning rate, or the step size. And $g^p(x)$ is the gradient on a data batch $x \in \mathcal{X}_p$, in which the $\mathcal{X}_p$ is a subset of dataset $\mathcal{X}$ computed by the local computing worker p.

$$w_{t+1} = w_t - \eta_t \frac{1}{P} \sum_{p=1}^{P} g^p(x)$$

Supported by the Open Fund from the State Key Laboratory of High Performance Computing (No. 201901-11).

X. Sun et al. (Eds.): ICAIS 2021, LNCS 12736, pp. 41–52, 2021.
https://doi.org/10.1007/978-3-030-78609-0_4

Traditional distributed SGD algorithms need to transfer all local gradients to relevant workers, which results in some communication overhead. As the model becomes larger, the communication overhead might be overwhelming in terms of training time. Many studies have proposed to address this issue. Two typical types include sparsification and quantization. Sparsification methods select a portion of gradients for transmission among them all and achieve a higher compression ratio. Differently, quantization methods use a low precision representation for the gradient. i.e., TernGrad [19] uses three values (−1, 0 and 1) and 1BitSGD [13] only uses 1 bit. The quantization could achieve a compression ratio of up to 32 at most, assuming the gradients are $Float32$ values.

In this paper, we proposed a novel algorithm named *Gaussian Averaging SGD* (GASGD), which is different from sparsification and quantization. Our algorithm GASGD only needs to transfer one 32-bit value for each worker and minimize the communication complexity. The key idea is based on an approximately Gaussian distribution of gradients, we maintain an average of the gradient distribution rather than all gradients from each worker. Our theoretical analysis shows that the GASGD can converge as other distributed algorithms do. Our empirical results also validate the analysis and that our algorithm has better performance than others. Respectively, GASGD has achieved 2.6× and 1.3× speedup on execution time per iteration compared to the QSGD and Top-K in training the LSTM-PTB model (which has nearly 66 million parameters).

Our Contributions as follows:

- By examining the scalability challenge of gradient synchronization in distributed SGD and analyzing its computation and communication complexities, we have proposed a Gaussian distribution averaging algorithm (GASGD) for distributed workers to exchange only one mean value globally.
- We give a theoretical analysis of the convergence of GASGD and demonstrated that our algorithm achieves an overall improvement compared to other sparsification and quantization algorithms with a lower computation complexity.

To the best of our knowledge, GASGD is the first attempt that can reduce the communication down to 32 bits per iteration for distributed SGDs.

2 Related Work

Quantization. 1-bit SGD [13] is to quantize the gradients into 1 bit to transfer, and the experiment in which it used the 1-bit SGD to train a speech model achieved a higher speedup. After 1-bit SGD, 8-bit quantization [6] has been proposed. It maps each $Float32$ gradient to 8 bit: 1 bit for sign,3 bit for the exponent, and the other 4 bit for the mantissa. Quantized SGD (QSGD) [2] proposed another method which used stochastic rounding to estimate the gradients and could quantize the gradients into 4 or 8 bit. Similar to QSGD, there is another approach named TernGrad, which uses 3-level values (−1, 0 and 1) to

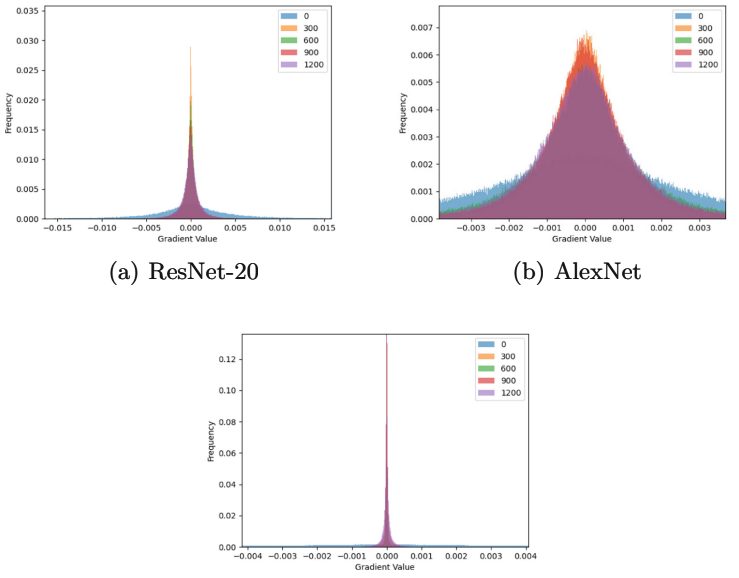
represent a gradient, could quantize the gradients into 2 bit. QSGD and TernGrad also give a theoretical analysis to demonstrate the convergence of their quantization algorithm. Moreover, both of them also do lots of experiments to demonstrate that their algorithm works a lot. SignSGD [3] transmits the sign of gradient elements by quantizing the harmful components to -1 and the others to 1. There are also some attempts to apply the quantization method to the entire model, not only the gradients to transfer, but also the parameters in the model. The DoReFa-Net [22] used 1-bit to represent the parameters in the model and 2-bit for gradients. Furthermore, LPC-SVRG [21] also applies that quantization method to another classical optimization algorithm - SVRG [9], which gives a code-based approach that combines gradient clipping with quantization.

Sparsification. Sparsifing the whole gradients to a part of them is another way to solve the communication bottleneck. Threshold-v [17] selects the elements by giving a threshold v that whose absolute values are larger than a fixed defined threshold value, which is difficult to choose in practice and uses zeros to represent the other elements. Unlike the threshold-v, Top-K [1] selects the k largest gradient values in absolute value, and there is also a random version named Random-K [16]. To further realize a lower loss on the accuracy, DGC [12] apply a local update inspired by the momentum SGD, and a warm-up step for the selection of hyperparameter K. After finding that the gradient variance affects the convergence rate, Wangni et al. [18] proposed an unbiased sparse coding to maximize sparsity and control the variance to ensure the convergence rate. Concurrently, Adacomp [5] has been proposed to automatically modify the compression rate depending on the local gradient activity, which realizes a 200× higher compression ratio for the fully-connected layer in deep learning model and 40× for the convolutional neural layer with a sightly loss in top-1 accuracy on the ImageNet.

3 Distributed SGD with Guassian Distribution Averaging

As mentioned in Sect. 1, since all workers are required to exchange their gradients, gradient synchronization poses a fundamental scalability challenge for data-parallel distributed SGDs. Although sparsity and quantization methods are significant to reduce the communication complexity of gradient synchronization, the computational efficiency of sparsity and quantization could be critical to gradient synchronization's scalability. Shi et al. pointed out that although the Top-K algorithm could reduce communication traffic for each worker, its computational overhead can offset communication reduction, which results in even higher execution time for each iteration. As we have observed in the experimental evaluation on distributed systems with a bandwidth of 100-Gbps bandwidth network, the high computational cost of the top-K algorithms can overshadow its benefits in communication efficiency. The same problem occurs with quantization algorithms, i.e., QSGD [2] and TernGrad [19]. Shi et al. [14] proposed a simple way to avoid expensive sorting and selecting the first K elements on all

gradient values. They assume a Gaussian distribution of gradients and estimates a statistical threshold to select the gradient value. It has proved the importance of low computation complexity for compression. All these studies above could reduce the computational cost of gradient decent while proving the model's ability to converge. However, they all require computing workers to exchange at least a not-so-small portion of gradients.



(a) ResNet-20 (b) AlexNet

(c) ResNet-110

Fig. 1. Progression of gradient distribution during training.

Based on previous studies, we proposed a novel approach to optimize the communication process. We assume that gradients computed per iteration obey a Gaussian distribution, and we could locally estimate standard deviation. Instead of selecting top gradient elements, we can exchange the standard deviation across the distributed workers and get a mean of these standard deviation values. A global mean of standard deviation could be used for gradient synchronization in all workers. We refer to our algorithm as *Gaussian Average* (GASGD). It effectively reduces the communication cost per iteration down to one values (32 bits), achieving a communication complexity of $\mathcal{O}(1)$. To further maintain the information, we also compute the mean of the gradient vector locally and adapt it to the decoding process.

3.1 Gradient Distribution

Shi et al. [14] have carried out many experiments to discuss the gradient distribution in deep learning models. Moreover, it assumed the Gaussian distribution of

gradient values and proposed the Gaussian-K sparsification algorithm. Figure 1 shows the frequency distribution of gradient values with a single machine for three representative models: ResNet-20 [8], AlexNet [11], and ResNet-110. We could find that most of the values are close on either side of zero, following a normal distribution. Besides, as the models finish more iterations of the training, more gradient values converge to the center around zero (a lower standard deviation than before). In our algorithm, we also give an assumption that the gradient obey a Gaussian distribution.

3.2 Details of GASGD

As we assume a Gaussian disrtibution of gradient values ($g \sim N(\mu, \sigma^2)$), for a gradient vector $\mathbf{g} = \{g_1, g_2, \ldots, g_n\} \in \mathbb{R}^n$, we could estimate two key parameters from all gradient values as follows,

$$\overline{\mu} = \mu(\mathbf{g}) = \frac{1}{n}\sum_{i=1}^{n} g_i, \quad \overline{\sigma} = \sigma(\mathbf{g}) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{i}(g_i - \overline{\mu})^2)}$$

Algorithm 1. Parallel GASGD algorithm on node k

Require: dataset $\mathcal{X}$
Require: minibatch size b per node
Require: the number of node N
Require: optimization function SGD
Require: init parameters ω_0, η_0
1: **for** $t = 0, 1, \ldots, T$ **do**
2: **for** $i = 1, \ldots, b$ **do**
3: Sample data x from the dataset $\mathcal{X}$
4: $g_t^t \leftarrow \frac{1}{b}\nabla f(x; w_t)$
5: **end for**
6: $\mu_k^t \leftarrow \mu(g_k^t)$ and $\sigma_k^t \leftarrow \sigma(g_k^t)$
7: $g_{normal}^t \leftarrow (g_k^t - \mu_k^t)/\sigma_k^t$
8: $\overline{\sigma}^t \leftarrow$ **Allreduce**$(\sigma_k^t,$ **average**$)$
9: $g_k^t \leftarrow g_{normal}^t \cdot \overline{\sigma}^t + \mu_k^t$
10: $\omega_{t+1} \leftarrow SGD(\eta_t, g_k^t)$
11: **end for**
12: $k_b \leftarrow Argmin(loss_k)$
13: $\omega^k \leftarrow Broadcast(\omega, k_b)$

The GASGD algorithm is described in details above. As shown in Algorithm 1, node k starts training with a learning rate of η_0 and an initial parameter ω_0. In an iteration t, node k compute its stochastic gradients g_k^t with a mini-batch form dataset $\mathcal{X}$ (Line 2 to 5). After obtaining local gradients, it then estimates the mean and standard deviation in Gaussian distribution through the equation above (Line 6). Before all nodes call the Allreduce operation to exchange their

local standard deviation and get back the global mean of standard deviations (Line 8), It would map their gradient value to a standard normal distribution (Line 7). Then the local gradient would be decode by using the global standard deviation (Line 9). Finally, the model's parameters are updated using the new gradient and the specific learning rate η_t with an optimization function SGD at Line 10. At the end of the training process, one more operation, for synchronizing the model across all nodes, is to find the node which has a minimal loss and broadcast the model parameters to the others.

3.3 Convergence Analysis

Distributed Optimization (specifically SGD) can be considered as an online learning system to be analyzed in its framework. The convergence analysis is based on the convergence proof of GOGA (General Online Gradient Algorithm) by Bottou [4], similarly as the previous works [19]. We adopt two assumptions and one lemma to our analysis.

Assumption 1. $L(\omega)$ has a single minimum ω^* and gradient $-\nabla_\omega L(\omega)$ always points to ω^*.

$$\forall \epsilon > 0, \quad \inf_{||\omega - \omega^*|| > \epsilon} (\omega - \omega^*)^T \nabla_\omega L(\omega) > 0$$

Assumption 2. Learning rate η_t is positive and constrained as follow,

$$\begin{cases} \sum \eta_t^2 < +\infty \\ \sum \eta_t = +\infty \end{cases}$$

The constraints about learning rate η_t ensure that η_t could change at a appropriate speed. We defined the square of distance between the current weight w and the minimun weight ω^* we want to get below:

$$h_t \triangleq ||\omega - \omega^*||^2$$

where $||\cdot||$ is l_2-norm. We also define the set of all random variables before step t as follows:

$$D_t \triangleq (z_{1\ldots t-1}, b_{1\ldots t-1})$$

Under the Assumption 1 and 2 above, using Lyapunov process and Quasi-Martingales convergence theorem, Bottou proved the Lemma 1 below.

Lemma 1. *If* $\exists \mathcal{A}, \mathcal{B} > 0$ *s.t.*

$$\mathbb{E}\{(h_{t+1} - (1 + \eta_t^2 \mathcal{B}) h_t) | D_t\} \leq -2\eta_t (w - w^*)^T \nabla_\omega C(\omega_t) + \eta_t^2 \mathcal{A}$$

then $L(z, \omega)$ *converges* ***almost surely*** *toward the minimum* ω^*. *i.e.* $P(\lim_{t \to +\infty} \omega_t = \omega^*) = 1$

To further give a proof to our algorithm, we make another assumption that the gradients have a bound as follows. We also denote that $g_t + \nabla G$ as the net gain. As $\mathbb{E}(\nabla G) = 0$, we would have $\nabla_\omega C(\omega_t) = \mathbb{E}(g_t + \nabla G)$.

Assumption 3 (Gradient Bound)

$$\mathbb{E}||g_t + \nabla G|| \leq \mathbb{A} + \mathbb{B}||\omega - \omega^*||^2$$

Theorem 1. *When the learning system is updated as follow equation,*

$$\omega_{t+1} = \omega_t - \eta_t(g_t + \nabla G)$$

then, it would ***converges almost surely*** *towards minimum* ω^*. *i.e.* $P(\lim_{t \to +\infty} \omega_t = \omega^*) = 1$

Proof.

$$h_{t+1} - h_t = -2\eta_t(\omega_t - \omega^*)^T(g_t + \nabla G) + \eta_t^2(g_t + \nabla G)^2$$

Taking the expectation of the equation above based on the condition D_t, we have

$$\mathbb{E}\{h_{t+1} - h_t | D_t\} = -2\eta_t(\omega_t - \omega^*)^T\mathbb{E}\{g_t + \nabla G | D_t\} + \eta_t^2\mathbb{E}\{||g_t + \nabla G||^2 | D_t\}$$

By using the following condition: $\nabla_\omega C(\omega_t) = g_t + \nabla G$, then

$$\begin{aligned} &\mathbb{E}\{h_{t+1} - h_t | D_t\} = -2\eta_t(\omega_t - \omega^*)^T\nabla_\omega C(\omega_t) + \eta_t^2\mathbb{E}\{||g_t + \nabla G||^2 | D_t\} \\ &\rightarrow \mathbb{E}\{h_{t+1} - h_t | D_t\} + 2\eta_t(\omega_t - \omega^*)^T\nabla_\omega C(\omega_t) = \eta_t^2\mathbb{E}\{||g_t + \nabla G||^2 | D_t\} \end{aligned}$$

From the Gradient Bound (Assumption 3), we can further have:

$$\mathbb{E}\{h_{t+1} - h_t | D_t\} + 2\eta_t(\omega_t - \omega^*)^T\nabla_\omega C(\omega_t) \leq \mathcal{A}\eta_t^2 + \mathcal{B}\eta_t^2||\omega - \omega^*||^2 = \mathcal{A}\eta_t^2 + \mathcal{B}\eta_t^2 h_t$$

$$\mathbb{E}\{(h_{t+1} - (1 + \eta_t^2\mathcal{B})h_t) | D_t\} \leq -2\eta_t(w - w^*)^T\nabla_\omega C(\omega_t) + \eta_t^2\mathcal{A}$$

The equation above satisfies the condition of Lemma 1, which is proved by Bottou, and could proves the Theorem 1.

4 Experiments and Results

In this section, we first describe our experimental setup and then present our evaluation results to validate the convergence of GASGD. Besides, we compare its performance with Dense SGD (without compression), one sparsification techniques Top-K [1] and one quantization technique QSGD [2].

Table 1. System parameters of each computation node.

HW/SW module	Description
CPU	Intel(R) Xeon(R)Gold 6132 14-core multi-core × 2
GPU	NVIDIA Tesla V100 SXM2 × 4
OS	CentOS Linux release 7.6.1810
Memory	256 GB(shared by 2 CPUs)
Development Environment	CUDA 10.0, PyTorch 1.3.0, Horovod 0.18.2
Network Between Nodes	100-Gbps InfiniBand

4.1 Experiment Setup

We performed all experiments on a GPU-V100 cluster. The main system parameters of each node in GPU-V100 are listed in the Table 1 above.

We have implemented GASGD on top of PyTorch [41] v1.3.0 with CUDA [44] v10.1 and utilized Horovod [45] v0.18.2 for data-parallel implementation of different models. Top-K and QSGD implementation are adapted from a Github repository (GRACE) [20], which is a gradient compression framework for distributed deep learning. Both implementations use the PyTorch Tensor API.

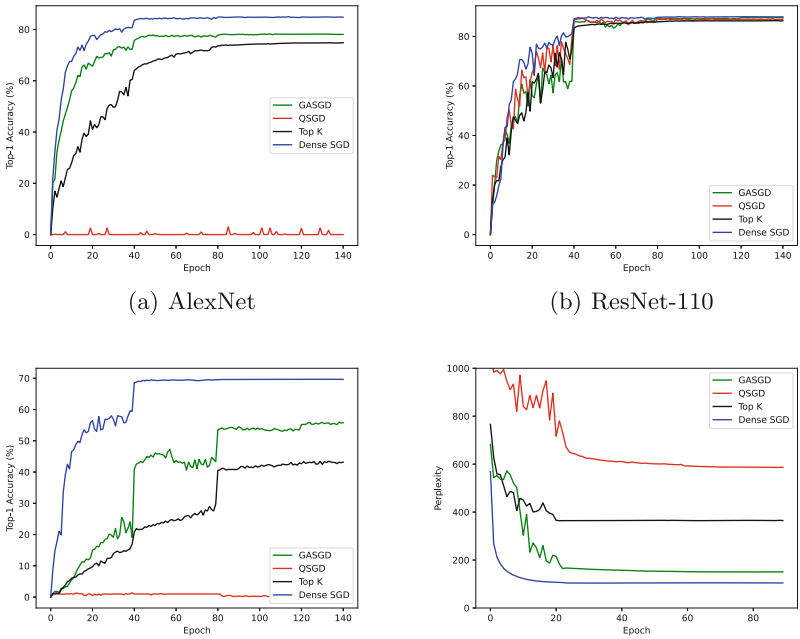
In our tests, we have employed two different DNN models, including (1) three types of Convolutional Neural Networks (CNNs), i.e., VGG-16 [15], ResNet-110 and AlexNet using CIFAR-10 [10] or CIFAR-100 dataset; and (2) RNN which consists of the recurrent neuron, i.e., the 2-layers Long Short Term Memory (LSTM) neural network model which has 1500 hidden units per layer using Penn Treebank corpus (PTB) dataset. We train the CNNs using the momentum SGD with learning rate decay and use vanilla SGD with learning rate decay for RNN.

Table 2. Experimental setup for neural network models

Type	Net	# Params	Dataset	Batch size	Base learning rate
CNN	AlexNet	23,272,266	CIFAR-10	128	0.01
	ResNet-110	1,727,962	CIFAR-10	128	0.01
	VGG-16	14,728,266	CIFAR-100	128	0.01
RNN	LSTM-PTB	66,034,000	PTB	20	22

4.2 Convergence Accuracy

To validate the convergence of the GASGD algorithm, we train all four deep learning models, with 140 epochs for AlexNet, ResNet-110, and VGG-16, 90 epochs for LSTM-PTB with four workers. We measure the top-1 accuracy for the first three models and the perplexity score for LSTM-PTB. The details are shown in Table 2.



(a) AlexNet (b) ResNet-110

(c) VGG-16 (d) LSTM-PTB

Fig. 2. Comparison of convergence accuracy with 4 workers.

Figure 2 shows that the convergence performance of different models with four workers. These results demonstrate that GASGD could converge and achieve a closer top-1 accuracy to Dense SGD within the same number of epochs among these three algorithms. GASGD achieves $78.27\%, 87.60\%, 56.46\%$ top-1 accuracy for AlexNet, ResNet-110, VGG-16 and 150.51 perplexity for LSTM-PTB (Dense SGD: $84.61\%, 87.99\%, 69.71\%$ top-1 accuracy and 104.83 perplexity). Furthermore, GASGD has a better performance than the other two algorithms for AlexNet, VGG-16, and LSTM-PTB.

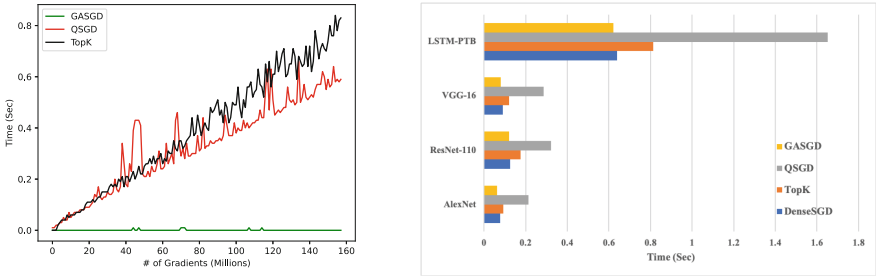
4.3 Computation and Communication Complexity

GASGD algorithm is designed to improve gradient synchronization with reduced communication traffic without costly additional computation to process the local gradients. To obtain an insight view on its impact to computation and communication in gradient synchronization, we have characterized the asymptotic computation complexity and the amount of communication traffic (number of bits) per worker for GASGD, in comparison with the dense SGD, QSGD, and Top-K. In data-parallel distributed SGD, each worker hosts a full copy of the model and the gradients after each training iteration. We assume a model with n parameters, therefore n gradients as well (Table 3).

Table 3. Comparison of computation and communication complexity

Compression method	Computation complexity	Communication cost # of bit
Dense SGD	$\mathcal{O}(1)$	$32n$
Top K	$\mathcal{O}(n + klogn)$	$32k$
QSGD	$\mathcal{O}(n^2)$	$2.8n + 32$
GASGD (ours)	$\mathcal{O}(n)$	32

Computation Complexity. For a gradient vector $g \in \mathbb{R}^n$, Dense SGD does not need to process gradient locally, and have a computation complexity of $\mathcal{O}(1)$. The top-K algorithm needs to sort and select the largest K value so that it has a computation complexity of $\mathcal{O}(n+klogn)$ with PyTorch implementation, where n is the complexity of sorting and $klogn$ for selecting. Moreover, QSGD computes the second norm (a complexity of $\mathcal{O}(n)$) and apply quantization for each gradient value. Thus it has a computation complexity of $\mathcal{O}(n^2)$ in total. GASGD has an overall computation complexity of $\mathcal{O}(n)$ because it just needs to compute one value for averaging, which is $\mathcal{O}(n)$.



(a) Comparison of GASGD computation time with other algorithm

(b) Comparison of Average Iteration Time with 4 Workers

Fig. 3. The performance of GASGD on computation and communication cost.

Communication Complexity. Obviously, Dense SGD transfers all the gradients for each worker with no information loss. Thus its cost is $32n$ bits. Top-K selects the k gradients for transferring, i.e., $32k$ bits. QSGD transfers $2.8n + 32$ bits, as the author reported. GASGD transfer one value for averaging (32 bits) and achieve a communication complexity of $\mathcal{O}(1)$ for each worker.

Based on the analysis above, we also have designed some experiments to validate our results. Figure 3(a) shows that GASGD has a much lower computation complexity than the Top-K and QSGD algorithm. As shown in Fig. 3(b), GASGD has a minimum average iteration time among all four algorithms. Besides, the

execution time per iteration is always the longest for the QSGD algorithm. Compare to the others, the main reason is that the overhead from its high computation complexity mitigates the benefit from the communication reduction on the 100-Gbps bandwidth InfiniBand network. These validation results demonstrate that GASGD has a better performance on convergence accuracy, computation complexity, and execution time with the other distributed algorithms. Furthermore, we could give a layer-wise implementation to improve algorithm performance over the initial implementation in which we stitch together the gradients of each layer.

5 Conclusion

In this paper, we proposed a novel compression GASGD method in distributed SGD based on the gradient distribution assumption, which just transfers one value for averaging and achieves a communication complexity of $\mathcal{O}(1)$ for each worker. We have theoretically analyzed the convergence of GASGD and also given the experimental results to validate our analysis. The GASGD has an overall improvement compared to the other algorithms. In the future, we would give a further optimization on both implementation and algorithm itself.

References

1. Aji, K.A.F., Heafield, K.: Sparse communication for distributed gradient descent. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, pp. 440–445 (2017). https://doi.org/10.18653/v1/d17-1045
2. Alistarh, D., Grubic, D., Li, J., Tomioka, R., Vojnovic, M.: QSGD: communication-efficient SGD via gradient quantization and encoding. In: Advances in Neural Information Processing Systems 30, pp. 1709–1720 (2017)
3. Bernstein, J., Wang, Y.X., Azizzadenesheli, K., Anandkumar, A.: signSGD: compressed optimisation for non-convex problems. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, vol. 80, pp. 560–569. Stockholmsmässan, Stockholm, 10–15 July 2018
4. Bottou, L.: Online learning and stochastic approximations (1998)
5. Chen, C., Choi, J., Brand, D., Agrawal, A., Zhang, W., Gopalakrishnan, K.: AdaComp: adaptive residual gradient compression for data-parallel distributed training. In: 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, pp. 2827–2835, January 2018
6. Dettmers, T.: 8-bit approximations for parallelism in deep learning. In: International Conference on Learning Representations (2016)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding (2018)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016). https://doi.org/10.1109/cvpr.2016.90
9. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. In: Advances in Neural Information Processing Systems 26, pp. 315–323 (2013)

10. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. University of Toronto, May 2012
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105 (2012)
12. Lin, Y., Han, S., Mao, H., Wang, Y., Dally, W.J.: Deep gradient compression: reducing the communication bandwidth for distributed training. In: International Conference on Learning Representations, May 2018
13. Seide, F., Fu, H., Droppo, J., Li, G., Yu, D.: 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In: Interspeech 2014 (2014)
14. Shi, S., Chu, X., Cheung, K.C., See, S.: Understanding top-k sparsification in distributed deep learning. In: International Conference on Learning Representations (2020)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations, May 2015
16. Stich, S.U., Cordonnier, J.B., Jaggi, M.: Sparsified SGD with memory. In: Advances in Neural Information Processing Systems 31, pp. 4447–4458 (2018)
17. Strom, N.: Scalable distributed DNN training using commodity GPU cloud computing. In: INTERSPEECH, pp. 1488–1492 (2015)
18. Wangni, J., Wang, J., Liu, J., Zhang, T.: Gradient sparsification for communication-efficient distributed optimization. In: Advances in Neural Information Processing Systems 31, pp. 1299–1309 (2018)
19. Wen, W., et al.: TernGrad: ternary gradients to reduce communication in distributed deep learning. In: Advances in Neural Information Processing Systems 30, pp. 1509–1519 (2017)
20. Xu, H., et al.: Compressed communication for distributed deep learning: survey and quantitative evaluation. Technical report, KAUST, April 2020
21. Yu, Y., Wu, J., Huang, J.: Exploring fast and communication-efficient algorithms in large-scale distributed networks. In: Proceedings of Machine Learning Research, vol. 89, pp. 674–683 (2019)
22. Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. In: IEEE Conference on Computer Vision and Pattern Recognition (2016)