Layer-wise Top-*k* Gradient Sparsification for Distributed Deep Learning

Guangyao Li
Institute of Automation, Chinese Academy of Sciences
Beijing, China
liguangyao2020@ia.ac.cn

***Abstract*—Distributed training is widely used in training large-scale deep learning models, and data parallelism is one of the dominant approaches. Data-parallel training has additional communication overhead, which might be the bottleneck of training system. Top-*k* sparsification is a successful technique to reduce the communication volume to break the bottleneck. However, top-*k* sparsification cannot be executed until backpropagation is completed, which disables the overlap of backpropagation computations and gradient communications, leading to limiting the system scaling efficiency. In this paper, we propose a new distributed optimization approach named LKGS-SGD, which combines synchronous SGD (S-SGD) with a novel layer-wise top-*k* sparsification algorithm (LKGS). The LKGS-SGD enables the overlap of computations and communications, and adapts to gradient exchange at layer-wise. Evaluations are conducted by real-world applications. Experimental results show that LKGS-SGD achieves similar convergence to dense S-SGD, while outperforming the original S-SGD and S-SGD with top-*k* sparsification.**

Keywords-component; deep learning; gradient sparsification; data parallelism

I. INTRODUCTION

Deep learning models are becoming larger which offers significant accuracy gain, however the computility and memory of a single GPU can no longer satisfy the requirements of training latest large-scale models. Therefore, distributed training that utilizing a large-scale computing system (such as GPU clusters) for training neural network models is a common practice[13]. Data parallelism is an extremely important strategy in distributed training. In data-parallel training, different nodes maintain consistent model parameters, while using different mini-batch samples in each training iteration. After calculating the local gradients on each node, the global gradients accumulated across all nodes through network communication. Finally, the global gradients will be applied for model parameters update.

Compared with single-node training, distributed training has additional communication overhead. After backpropagation, model parameters or gradients need to be exchanged with other nodes. Due to the large communication volume, the computing node should wait for a long time for gradient exchange, which become the main bottleneck of distributed training. In order to break the bottleneck caused by communication, we should reduce the communication volume of data-parallel training[2][3][6][7][8][14]. Gradient sparsification[1][5][10][14] is a promise approach to significantly reduce the communication volume.

The key idea of gradient sparsification is that, in deep learning model training, not all gradient values are of equal importance, and usually up to 99% of the gradient values contribute few for model convergence. In other words, only 1% important gradient values are accumulated across all nodes to obtain the global gradients.

Top-*k* sparsification[14] is a successful gradient sparsification approach, each node transfers only the *k* largest (in terms of the absolute value) of its gradient values. The convergence of top-*k* sparsification has been proved in theory and practice[1][14], however, it suffers from an issue that hinders its application: the selection of the top-*k* value is based on the gradients of entire model, so the sparsification and communication cannot start until backpropagation is completed. In many distributed training frameworks, the gradient exchange is executed at a layer-wise[11]. Once the gradients of a layer have been calculated, it will be exchanged among nodes immediately, so that the gradient communications and back-propagation computations overlap, reducing communication overhead. However, top-*k* gradient sparsification can only be performed after backpropagation, cannot overlap the computations and communications, which limits the system scaling efficiency. To conquer this issue, we propose a new gradient sparsification algorithm named layer-wise top-*k* gradient sparsification (LKGS).

We implement LKGS in PyTorch and compare it to original S-SGD and S-SGD with top-*k* sparsification. We evaluate it on ResNet50[5] and VGG16[9] to verify the convergence and efficiency. Our contributions are as follows:

- We propose a layer-wise sparsification algorithm to enable the overlap of backpropagation computations and gradient communications.
- We implement the proposed LKGS-SGD algorithm atop PyTorch and MPI, which achieves improvement without losing convergence.

II. PRELIMINARIES

A. Mini-batch SGD

In deep learning model training, the loss function is used to measure the difference between ground truth and the output predictions of model. The training process of deep learning

model is the process of adjusting model parameters to reduce the loss. Stochastic gradient descent (SGD) is the most used optimization algorithm in deep learning, and it is used in conjunction with the backpropagation algorithm to update model parameters.

Let m be the size of a mini-batch, w_t the parameters of the neural network model at the t-th iteration, (x_i, y_i) a sample and its label in the batch, and L the loss function. Each round of training has two stages. In forward propagation, calculating loss by loss function L with current model parameters and samples in mini-batch. In backpropagation, use the backpropagation algorithm to calculate the gradients of all parameters:

$$G_t(w_t) = \frac{1}{m}\sum_{i=0}^{m} \nabla L(w_t, x_i, y_i) \quad (1)$$

and use $w_{t+1} = w_t - \alpha G_t$ to update the model parameters, α is the learning rate. In single-node training, forward propagation and backpropagation are the main time cost, so the total time of one iteration can be approximated by t_f+t_b.

Algorithm 1 S-SGD with top-k sparsification

Inputs: dataset D, initialized weights w, mini-batch size m, iterations T, learning rate α, the number of worker P, the number of gradient values to select k.

1: Initialize $G_0^i = 0$
2: **for** $t = 1 \rightarrow T$ **do**
3: Sampling a mini-batch of data D_t^i from D;
4: $G_t^i = G_{t-1}^i + \nabla L(w_t, D_t^i)$
5: Select threshold thr = the k^{th} largest value of $|G_t^i|$;
6: $Mask = |G_t^i| > thr$;
7: $G_{local}^i = G_t^i \odot Mask$;
8: $G_t^i = G_t^i \odot \neg Mask$; // The residuals of gradients
9: G_{global}^i =TopKAllReduce(G_{local}^i)
10: $w_{t+1} = w_t - \alpha G_{global}^i$
11: **end for**

B. Synchronous SGD

Synchronous SGD (S-SGD) is widely used in data parallelism. It is mainly to expand the SGD to adapt to the distributed environment. After each node calculates its gradients locally, it exchanges gradients with other nodes through the AllReduce[4] operation, accumulates global gradients and applies them to model parameters updates. The update formula of parameters is:

$$w_{t+1} = w_t - \alpha \sum_{i=1}^{P} G_i(w_t) \quad (2)$$

where P is the number of computing system nodes. S-SGD keeps the model parameters consistent on different nodes, which is equivalent to increasing the mini-batch size by P times. Since local gradients are distributed across different nodes, gradients exchange involves communication overhead. The total time of one iteration can be approximated by $t_f+t_b+t_c$.

In model training, the size of the local gradients and global gradients are the same as the model trainable parameters. Suppose the cluster has P nodes, the size of trainable parameters is N, and the gradient exchange through Ring-AllReduce algorithm, leading to $2N(P-1)/P$ communication volume. When the model size is large, due to the limitation of bandwidth, t_c far exceeds t_f+t_b, computing resources are idle most of the time, the speedup of data-parallel is extremely low, and communication overhead becomes the bottleneck of model training. For reducing the total training time, S-SGD usually uses computation-communication overlap technique, as is shown in Figure 1.

C. Top-k sparsification

Top-k sparsification is a promise approach to reduce the communication volume[14]. The key idea is that the gradient with a larger absolute value have a greater impact on model convergence, so each node only transfers the k largest gradient values (in terms of the absolute value). The rest of gradients will be added to the local gradients in the next iteration. Global gradients are obtained by accumulating local sparse gradients.

S-SGD with top-k sparsification has additional sparsification overhead. It significantly reduces the communication volume, but due to its dependence on entire gradients, cannot overlap the backpropagation computations and gradient communications, as is shown in Figure 1. . The total time of one iteration can be approximated by $t_f+t_b+t_c+t_s$.

The pseudo-code of top-k sparsification S-SGD is shown in Algorithm 1.

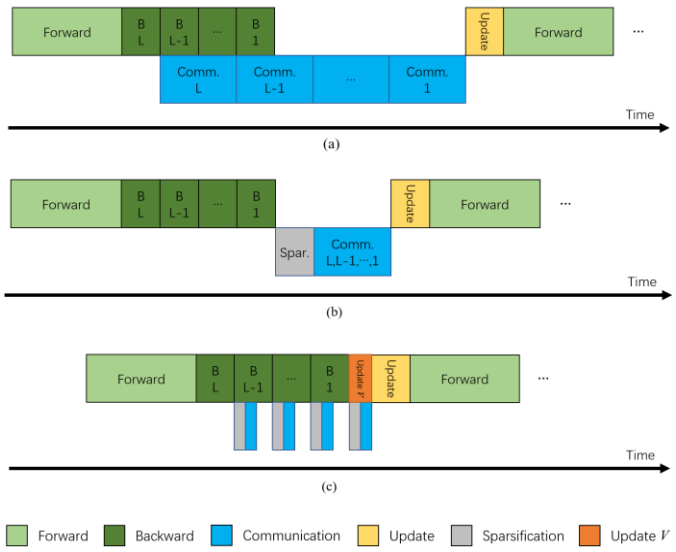

Figure 1. Comparison between thress distributed training algorithms: (a) the basic data parallelism with computations and communication overlap (Dense S-SGD), (b) the data parallelism with top-k sparsification, and (c) our proposed layer-wise top-k gradient sparsification algorithm.

III. METHOD

The layer-wise top-k algorithm requires the top-k gradient values in current layer to be selected and exchanged immediately after calculating the gradients, but at this time the entire gradient

values of model have not been generated, and we can only know the calculated partial model gradients. The real top-k gradients are based on the complete gradients, so the ideal layer-wise top-k algorithm is difficult to achieve. Our layer-wise top-k algorithm relies on estimated number of to top-k gradients in each neural network layer to approximately select the top-k values, thereby avoiding the dependence on the complete gradients. How to accurately estimate the number of top-k gradients contained in each neural network layer is the key issue of layer-wise top-k.

The main idea of our algorithm is to regard the distribution of gradient values during training as a slow changing stochastic process *G(t)*. Specifically, the statistics of *G(t)*, *G(t+1)* change very slowly. Therefore, we can use the information of the previous iteration to estimate the that of in this iteration and determine the number of top-k gradient values in each network layer immediately. We utilize a vector $V = [sub\_k_0, sub\_k_1, ..., sub\_k_l]$ ($\sum sub\_k_i = k$) to storage the number of top-k gradient values of each neural network layer and initialize it with the gradients size of each layer at the beginning, that is, in the first iteration nodes exchange all gradient values. For each iteration of training, after the gradients of layer i are calculated, its top-sub_k_i gradient values are selected and exchanged with other nodes. And after the model backpropagation is completed, nodes calculate the top-k threshold and update V. The communication of the gradients exchange will overlap with the gradient calculation of shallower model layers, thus hiding a certain amount of communication overhead. It should be noted that since the sparsification requires GPU computing resources, the overall sparsification overhead will not be reduced.

We validated our claim through an experimental result from different deep learning models. We take the sum of the absolute values of top-k gradients as its influence on the model update and compare the real top-k gradients and the layer-wise top-k gradients. The average difference of two algorithm is only 0.8%, which is extremely low. The impact of the layer-wise top-k algorithm on model update is similar to that of the real top-k. Compared with current top-k algorithms, our LKGS-SGD overlaps the computations and communications, which increase the training scaling efficiency.

The behavior of our LKGS-SGD is shown in Figure 1. , and the pseudo-code is shown in Algorithm 2.

Algorithm 2 S-SGD with layer-wise top-k sparsification

Inputs: dataset D, initialized weights w, mini-batch size m, iterations T, learning rate α, the number of worker P, the number of gradients to select k, the number of layer l.

1: Initialize G_0^i, V
2: **for** $t = 1 \rightarrow T$ **do**
3: Sampling a mini-batch of data D_t^i from D;
4: Feed-forward computation;
5: **for** j = $l \rightarrow 1$ **do**
6: $G_t^{i,j} = G_{t-1}^{i,j} + \nabla L(w_t, D_t^i)^j$;
7: Select threshold thr = the $sub\_k_j^{th}$ largest value of $|G_t^{i,l}|$;
8: $Mask = |G_t^{i,j}| > thr$;
9: $G_{local}^{i,j} = G_t^{i,j} \odot Mask$;
10: $G_t^{i,j} = G_t^{i,j} \odot \neg Mask$; // The residuals of gradients
11: $G_{global}^{i,j} = \text{AllReduce}(G_t^{i,j})$;
12: **end for**
13: update V
14: $w_{t+1} = w_t - \alpha G_{global}^i$
15: **end for**

IV. Experiments

A. Setup

Our experiments executed on 4 compute nodes. Each node has an Intel Xeon Gold 5117 CPU with 32GB of RAM, and one NVIDIA RTX2080ti GPU with 12GB global memory. The machines are connected by a 10Gbps Ethernet interface. Machines run 64-bit Ubuntu 18.04 with CUDA toolkit 11.2. The deep learning framework is PyTorch at version 1.7.0. The communication library is mpi4py, which built with MPICH 3.3.2.

We use VGG16 with 134M parameters on and ResNet50 with 24M parameters. The optimizer is SGD with learning rate of 0.01 for VGG16, and 0.001 for ResNet50. the mini-batch size is 64 per machine. We utilize the top-k function provided by PyTorch.

B. Convergence

In much previous work[1][12][14], the convergence of top-k S-SGD has been verified, so we do not include the convergence experiments of the top-k algorithm.

We use 1%, 5%, and 10% densities to analyze the convergence of our algorithm. The models are training on cifar-10 dataset. Density is defined as k/N.

The convergence of VGG16 and ResNet50 is shown in Figure 2. The results show that the convergence curve of the LKGS-SGD is almost the same as that of the basic data parallelism at different densities, even for ResNet50, LKGS-SGD converges slightly better than the baseline.

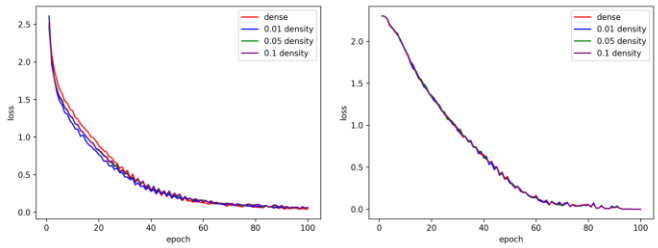

Figure 2. The convergence performance of ResNet50 and VGG16 with 1%, 5% and 10% density. **Left**: the result of ResNet50. **Right**: the result of VGG16.

C. *Time performance analysis*

We also use 1%, 5% and 10% density to analyze the performance of the LKGS-SGD. The ResNet50 is trained on cifar-10 dataset and VGG16 is trained on Caltech101 dataset.

We break down the time of an iteration into three parts: forward and backward pass time, sparsification time, and communication time. Since LKGS-SGD overlap the computations and communications, its communication time represents the time between the end of backpropagation and that of gradient exchange. The result is shown in the Figure 3.

Due to the high communication volume of dense AllReduce, the communication time is much longer than the computation time, which becomes the bottleneck of training, and the total time of an iteration is longer than that of other training algorithms. Our proposed algorithm performs similarly on two models, and the communication time is significantly lower than the top-*k* sparsification, For ResNet50, the communication time of our algorithm is reduced by 67% on average, and the communication overhead is almost completely hidden at the density 1%. For VGG16, communication time is reduced by 69% on average. However, compared with the top-*k* sparsification, the total time of an iteration is not significantly reduced, only reduced by 3.5%, and even at 1% density, the total time increases slightly. The reason is that the top-*k* function is called at each layer leads to a significant increase in the sparsification overhead. Therefore, it is worthy to explore more efficient selection algorithms for gradient sparsification.

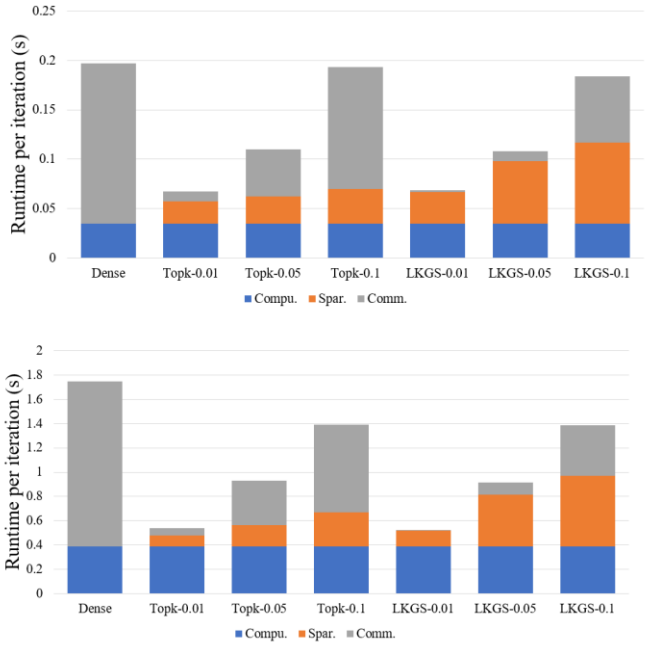

Figure 3. Time breakdown of computation, sparsification and communication. "Compu." indicates forward and backward computation time, "Spar." indicates sparsification time, and "Comm." indicates the gradient communication time. 0.01, 0.05 and 0.1 indicates the density of 1%, 5% and 10%. **Above**: the result of ResNet50. **Below**: the result of VGG16.

V. CONCLUSION

In this paper, we propose a novel layer-wise top-*k* sparsification algorithm to speedup gradient exchange. Our proposed algorithm enables the overlap of backpropagation computations and gradient communications by pre-estimate the number of top-*k* values at each neural network layer. Compared with top-*k* sparsification, it offers a higher scalability for data parallelism. The experimental results on real-world applications show that our algorithm reduces communication time without impacting the convergence. In future work, we would like to further investigate more efficient algorithms to reduce the sparsification overhead.

REFERENCES

[1] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in The 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 440–445.

[2] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp: Adaptive residual gradient compression for data parallel distributed training," in The 32nd AAAI Conference on Artificial Intelligence, 2018.

[3] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in Advances in Neural Information Processing Systems, 2018, pp. 5973–5983.

[4] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," Concurrency and Computation: Practice and Experience, vol. 19, no. 13, pp. 1749-1783, 2007.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770-778.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," The Journal of Machine Learning Research, vol. 18, no. 1, pp. 6869–6898, 2017.

[7] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "SIGNSGD: Compressed optimisation for non-convex problems," in International Conference on Machine Learning, 2018, pp. 559–568.

[8] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized SGD and its applications to large-scale distributed optimization," in International Conference on Machine Learning, 2018, pp. 5325-5333.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[10] P. Jiang and G. Agrawal, "A linear speedup analysis of distributed deep learning with sparse and quantized communication," in Advances in Neural Information Processing Systems, 2018, pp. 2530–2541.

[11] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," arXiv preprint arXiv:1802.05799, 2018.

[12] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in Advances in Neural Information Processing Systems, 2018, pp. 4452–4463.

[13] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," ACM Computing Surveys (CSUR), vol. 52, no. 4, pp. 1-43, 2019.

[14] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in International Conference on Learning Representations, 2018.