GradZip: Gradient Compression using Alternating Matrix Factorization for Large-scale Deep Learning

Minsik Cho[1] **Vinod Muthusamy**[2] **Brad Nemanich**[1] **Ruchir Puri**[2]
[1]IBM Systems, Austin, TX, USA [2]IBM Research, Yorktown Heights, NY, USA
minsikcho@us.ibm.com

Abstract

We present a novel gradient compression algorithm, *GradZip*, to accelerate distributed deep learning by minimizing the communication overhead. Unlike the popular sparsification-based compression techniques, GradZip performs dense compression based on alternating matrix-decomposition tailored for deep learning. Therefore, the compressed gradient can be directly communicated and reduced through the standard collective *all-reduce* available in most high-performance deep learning frameworks. The key insight is that all the learners will have a synchronized tensor from the last *all-reduce* in training which can be exploited to obtain high-quality gradient factorization very efficiently in the subsequent iteration. When GradZip used with an industry-leading *all-reduce*, it delivers significant compression performance: $100\times$ compression and $5\times$ training speedup with small accuracy loss of 0.22% for ResNet50 with half-precision quantization.

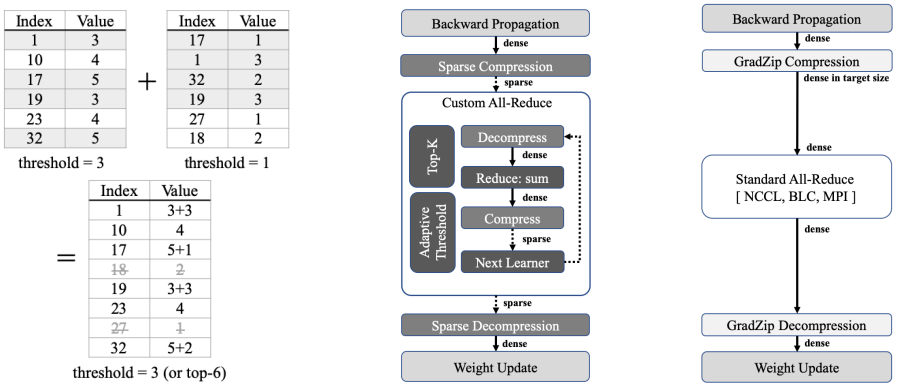
1 Introduction

One approach to address the communication bottleneck in distributed deep learning is a faster synchronization algorithm, such as widely used collective *all-reduce* operations [33, 4, 13, 34, 8, 17, 27]. The other approach is a compression algorithm to reduce the traffic volume by quantization and/or sparsification [5, 23, 37, 3, 38, 32]. Although the two approaches mentioned above have the same goal, they cannot be easily integrated into one effective solution because the two are designed to work on different data representations. All real-world collective operations are designed for dense computation, while quantization followed by sparsification relies on sparse representation [16].

Consider in Fig. 1 (a) where two sparsely represented and already compressed gradients from different learners (on the left) to be reduced during *all-reduce* where each element is denoted as a pair of indices and values (which is a generalization of sparse-encoding). Then, we can see the following issues: **a)** It is complex to map the scattered index-value pairs across two compressed representations [31], as the row-by-row operation is not permissible. Therefore, it is required to check and locate its position if necessary. For instance, while $(1, 3)$ from both can be reduced directly, $(10, 4)$ has no counterpart to be added and $(17, 5)$ has one, but on a different row. **b)** It is required to handle the overflow after reduction. As on the bottom in Fig. 1 (a), it is likely that the size of the initial reduction is larger than the contracted *all-reduce* size (which is 6 in this example). Thus, either a thresholding scheme or a top-K selection is needed [12, 31, 2, 23]. For example, $(18, 2)$ and $(27, 1)$ need to be dropped with a threshold 3 or top-6 selection. **c)** When a top-K algorithm with a linear complexity is used [23, 31], matching indices would be harder, because the outcome is not guaranteed to be sorted.

Such incompatibility issues arise from the fact that the collective operation standard [25] is defined to handle dense data representation and becomes a dominating data-parallel back-end [25, 27, 8, 34, 17, 13], while sparse gradient compression is popularly researched due to the inherent sparsity of gradient tensors [5, 23] and has been targeting a parameter-server architecture [5, 22]. To address such challenge, we propose GradZip, which is based on alternating matrix factorization enabled by

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.



(a) Reducing two sparse tensors in *all-reduce*.

(b) Sparse compression for *all-reduce*

(c) GradZip for *all-reduce*

Figure 1: Comparison between sparse gradient compression and GradZip for *all-reduce*. Solid and dotted lines represent dense and sparse formats, respectively.

the nature of distributed deep learning. GradZip utilizes the factorization results from the previous synchronization to deliver high-quality gradient factorization for the current synchronization. The matrix factorization inherently outputs a compressed gradient in dense representation, hence the compatibility issue between compression and*all-reduce* no longer exists.

2 GradZip

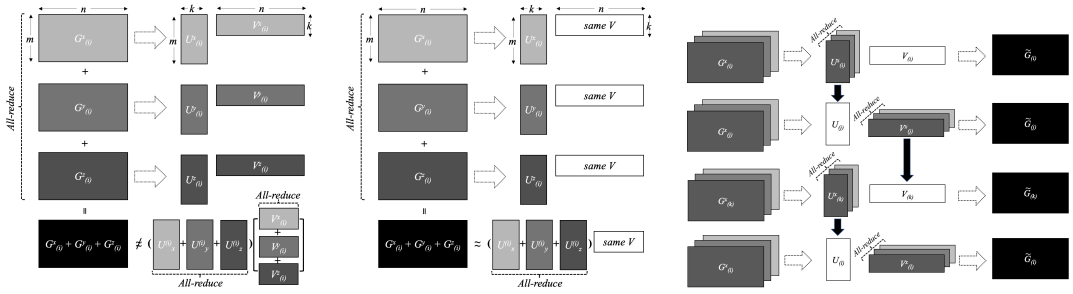
The key idea in GradZip is to compress a gradient tensor into a dense representation so that the compressed output can be directly aggregated using the standard *all-reduce*. Fig. 1 (b) and (c) highlight the key benefits of GradZip over sparse gradient compression for *all-reduce*. As discussed in Section 1, using sparse representation along with *all-reduce* is complicated as shown in Fig. 1 (b), where expensive temporary format conversion (i.e., decompress-reduce-compress) steps are repeated in *every* reduction along with ad-hoc threshold knobs and/or top-K selection [22, 12]. On the other hand, GradZip in Fig. 1 (c) compresses the gradient into a size-controlled dense format which is readily reducible with any standard *all-reduce* libraries. Such simplicity will greatly promote its adoption and integration into deep learning frameworks [28, 1, 18, 26, 6] and preserve the performance benefits of highly-optimized *all-reduce* implementations [34, 27, 8].

GradZip is based on matrix-factorization where a data representation is factorized into a set of much smaller ones [9, 14, 24, 21]. For a learner x in data-parallel training, factorizing a gradient tensor $G^x_{(i)}$ at (i)-th iteration can be described as follows:

$$\min_{U^x_{(i)}, V^x_{(i)}} \|G^x_{(i)} - U^x_{(i)} V^x_{(i)}\|^2_F + \lambda(\|U^x_{(i)}\|^2_F + \|V^x_{(i)}\|^2_F) \tag{1}$$

where $G^x_{(i)} \in R^{m \times n}, U^x_{(i)} \in R^{m \times k}$, $V^x_{(i)} \in R^{k \times n}$, and k is a latent variable that captures the underlying low-rank structure in $G^x_{(i)}$ and a hyper-parameter to tune the degree of compression at the same time. Gradient compression based on matrix factorization can have the following benefits: **a)** The factorization results are in dense representation which can be directly used for collective operations such as *all-reduce*. **b)** The output size is exactly controlled by a desired k, unlike sparsification where additional steps are needed (i.e., top-K selection or thresholding). **c)** The computation can be done by dense BLAS which can be highly efficient on GPUs where training is already running [10, 7]. Even though the benefits from dense representation are obvious, solving Eq. (1) for the gradient compression purpose is not trivial and faces the following two challenges: **a)** Eq. (1) is a non-convex optimization problem, thus computationally too expensive to be solved for gradient compression. **b)** Even if Eq. (1) could be instantly optimized out, the communication pattern to utilize $U^x_{(i)}$ and $V^x_{(i)}$ is not scalable: *allgather-broadcast* (i.e., parameter server) [35] or *all-to-all*.

Fig. 2 (b) elaborates the second challenge where the gradient tensors are factorized in the three learners $L = \{x, y, z\}$. After obtaining high-quality factorization results, computing $\sum_{x \in L} U^x_{(i)}$ and $\sum_{x \in L} V^x_{(i)}$ through *all-reduce* is not helpful, as there is no easy way to reconstruct the approximated $\sum_{x \in L} G^x_{(i)}$ with such aggregated results. In fact, it becomes necessary to share each $U^x_{(i)}$ and $V^x_{(i)}$



(a) *all-reduce* unfriendly naive gradient factorization.

(b) *all-reduce* compatible gradient factorization in GradZip.

(c) *all-reduce* compatible alternating gradient factorization in GradZip.

Figure 2: GradZip to accelerate *all-reduce* with the universally same matrix across all the learners.

individually within L so that each learner can compute $\sum_{x \in L} U^x_{(i)} V^x_{(i)}$ independently, which is very inefficient and costly. We observed that the above challenges can be nicely addressed if either $V^x_{(i)}$ or $U^x_{(i)}$ is fixed and *somehow identical* in L, which is depicted in Fig. 2 (b).

- Eq. (1) is a convex problem [14, 29, 19], once either matrix is fixed. Thus, the other matrix can be computed optimally by solving the following linear systems. With a fixed and identical $V^x_{(i)}$ assumed for simplicity, $U^x_{(i)}$ can be obtained by solving $(V^x_{(i)} V^{x\intercal}_{(i)} + \lambda I)\boldsymbol{U}^{x\intercal}_{(i)} = V^x_{(i)} G^{x\intercal}_{(i)}$ or $(U^{x\intercal}_{(i)} U^x_{(i)} + \lambda I)\boldsymbol{V}^x_{(i)} = U^{x\intercal}_{(i)} G^x_{(i)}$ where λ is a parameter to regularize $U^x_{(i)}$. Fig. 2 (b) illustrates the case where 3 learners, $L = \{x, y, z\}$ are factorizing $G^{\{x,y,z\}}_{(i)}$ in parallel with a fixed and identical V in grey.
- With $V^x_{(i)}$ being identical across L, only $\sum_{x \in L} U^x_{(i)}$ is required to reconstruct $\sum_{x \in L} G^x_{(i)}$ which is naturally mapped to *all-reduce* as in Fig. 2 (b). As $\sum_{x \in L} U^x_{(i)}$ incurs less network traffic than $\sum_{x \in L} G^x_{(i)}$, the proposed matrix factorization for gradient compression will accelerate distributed SGD [36, 3, 23, 5].

To ensure the same $V^x_{(i)}$ across L, GradZip reuses the last *all-reduce* result as a fixed $\tilde{V}$ as shown in Fig. 2 (c), which leads to alternating gradient factorization. In distributed SGD, the gradients could be highly correlated in a sense that successive gradients are laid in a proximity with similar characteristics like curvatures [30]. In that sense, it is very probable that the previous factorization result could be helpful in solving the future factorization problems. It would be possible to use a moving average scheme instead of simply using the last one as well, but at the cost of a larger memory footprint. Finally, for a given hyper-parameter k, the compression rate is $\frac{uncompressed\_size}{compressed\_size} = \frac{2mn}{(m+n)k}$.

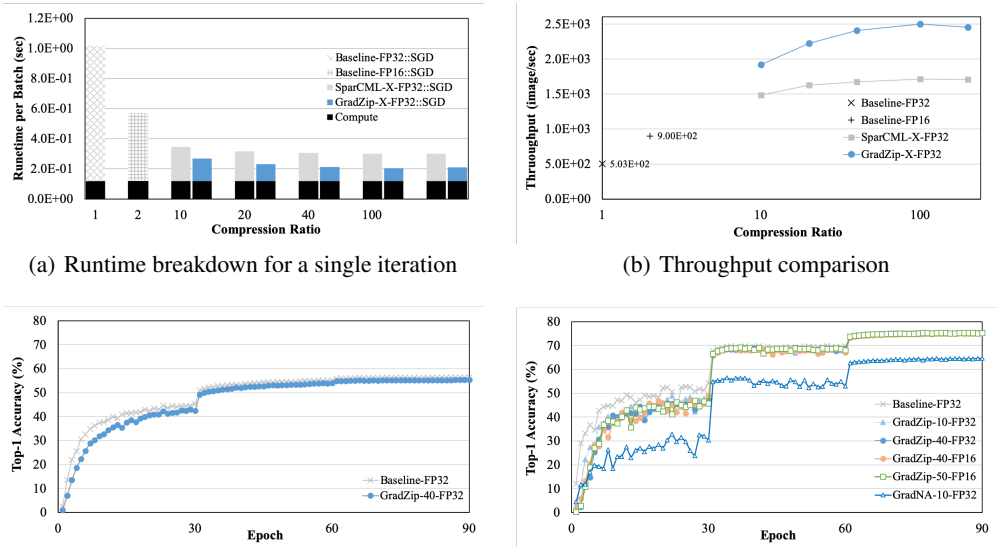
3 Experimental Results

We implemented GradZip for GPU in C++ with a GPU BLAS library [10] and integrated into PyTorch 1.1 [28]. We used eight linux nodes in a cloud platform for our experiments. Each node has two Nvidia V100 GPUs and networked via 10 Gbps Ethernet. For *all-reduce*, we used the latest NCCL2 [27]. All the tensors including gradients are computed in single-precision (FP32). We experimented the *all-reduce* ready compression techniques in Table 1. We did not explore other lower-bit quantization techniques except half-precision (FP16) as the latest NCCL2 and GPU do not support any non-standard format. With FP16, all schemes offer $2\times$ additional bit-wise compression.We trained AlexNet and ResNet50 [20, 15] with ImageNet1K [11] to evaluate different compression schemes for *all-reduce*. Each training job ran for 90 epochs with a mini batch-size of 32 per GPU [23, 3]. The initial learning rate was scaled down by $10\times$ every 30 epochs, following the standard practice. The momentum and the weight-decay were set as 0.9 and 1E-04, respectively.

Fig. 3 (a) and (b) show the performance and training throughput of training instances with different compression schemes. In Fig. 3 (a) runtime is separated into compute and SGD for a single iteration (i.e., in-between *weight update*) of ResNet50 training. The per-iteration computation time is independent of the number of learners, so we extracted the pure computation time by running a training job on a single GPU for the given batch size (which is about 0.118 sec). The SGD time is obtained by

Baseline-FP32	Gradients communicated without any compression or quantization.
Baseline-FP16	Gradients only quantized into half-precision (16bit) using CUDA native support.
SparCML-X-FP23	Sparse tensor *all-reduce* based on non-zero gradient indices overlap checking with X target compression ratio similar to [31].
GradZip-X-FP32	GradZip with X target compression ratio with a regularization, $\lambda = 1e-5$.
GradZip-X-FP16	**GradZip-X-FP32** with further quantization into half-precision (16bit).
GradNA-X-FP32	Same as **GradZip-X-FP32** except alternating

Table 1: Algorithms evaluated in this paper.



(a) Runtime breakdown for a single iteration

(b) Throughput comparison

(c) AlexNet

(d) ResNet50

Figure 3: GradZip performance results on ImageNet1K training.

subtracting the single GPU compute time from the per-iteration time. Therefore, the SGD measures include all synchronization overheads, including extra memory operations and imbalance among the GPUs and node (which we refer to as jitter).

- Fig. 3 (a) clearly shows that compression reduces the SGD overhead: both Baseline-FP16, SparCML, and GradZip all accelerated training to varying degrees. However, the overall speedup gets saturated once it passes 40× compression due to Amdahl's law, where **GradZip-40-FP32** is 2× faster than **SparCML-40-FP32**. The overhead in **SparCML-40-FP32** mainly comes from the need to compare indices to detect overlaps before the reduction.
- Fig. 3 (b) shows throughput improvements due to faster per-iteration-time. Again, we observed speedup saturation around 40× compression, and **GradZip-40-FP32** delivers 4.8× speedup over the **Baseline-FP32**.

Based on the throughput results, we trained AlexNet and ResNet50 with mainly a 40× compression target to study the impacts of compression on the final test accuracy as shown in Fig. 3 (c) and (d).

- For ResNet50 in Fig. 3 (d), **Baseline-FP32** yielded a 75.49% accuracy while **GradZip-40-FP32** delivered 75.17%. Lowering the compression ratio to 10× with **GradZip-10-FP32** only marginally improved the accuracy to 75.19%. However, with **GradNA-10-FP32**, the training barely converges to 64.65%, reconfirming the effectiveness of alternating gradient factorization.
- **GradZip-50-FP16** (100× end-to-end compression) offered negligible accuracy drop from **Baseline-FP32** (0.22%), confirming GradZip's compatibility with quantization schemes.
- Across all the tests, we were able to see the gradient staleness effect in the first 30 epochs as observed by [5], which can be mitigated by known techniques [23].

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. *CoRR*, abs/1704.05021, 2017.

[3] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[4] Baidu. https://github.com/baidu-research/baidu-allreduce. 2017.

[5] C. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan. Adacomp : Adaptive residual gradient compression for data-parallel distributed training. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, 2018.

[6] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[7] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[8] M. Cho, U. Finkler, and D. Kung. BlueConnect: Novel Hierarchical All-Reduce on Multi-tired Network for Deep Learning. In *Proceedings of the Conference on Systems and Machine Learning (SysML)*, 2019.

[9] J. Chung and T. Shin. Simplifying deep neural networks for neuromorphic architectures. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, 2016.

[10] cuBLAS. http://docs.nvidia.com/cuda/cublas.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[12] N. Dryden, S. A. Jacobs, T. Moon, and B. Van Essen. Communication quantization for data-parallel training of deep neural networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, MLHPC '16, 2016.

[13] Facebook. https://github.com/facebookincubator/gloo.

[14] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *J. Mach. Learn. Res.*, 16(1), Jan. 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[16] ICRL18 openreview.net for Deep Gradient Compression. https://openreview.net/forum?id=SkhQHMW0WnoteId=ryR$_PyE-f$.

[17] 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, and X. Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. 2018.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[19] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), Aug. 2009.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[21] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, 2000.

[22] H. Lim, D. Andersen, and M. Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. In *Proceedings of the Conference on Systems and Machine Learning (SysML)*, 2019.

[23] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations, ICLR 2018*, 2018.

[24] Z. Lu, V. Sindhwani, and T. Sainath. Learning compact recurrent neural networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2016*, 2016.

[25] MPI Forum. https://www.mpi-forum.org. 2018.

[26] Y. Niitani, T. Ogawa, S. Saito, and M. Saito. Chainercv: a library for deep learning in computer vision. *CoRR*, abs/1708.08169, 2017.

[27] NVidia. https://developer.nvidia.com/nccl. 2017.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[29] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, 2010.

[30] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 1999.

[31] C. Renggli, D. Alistarh, and T. Hoefler. Sparcml: High-performance sparse communication for machine learning. *CoRR*, abs/1802.08021, 2018.

[32] N. Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *INTERSPEECH*, 2015.

[33] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, Feb. 2005.
[34] Uber. https://eng.uber.com/horovod. 2017.
[35] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, and D. Papailiopoulos. Atomo: Communication-efficient learning via atomic sparsification. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, 2018.
[36] J. Wang and G. Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. 2018.
[37] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for communication-efficient distributed optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*. 2018.
[38] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 2017.