IntML: Natural Compression for Distributed Deep Learning

Samuel Horváth Chen-Yu Ho Ludovít Horváth
Atal Narayan Sahu Marco Canini Peter Richtárik
KAUST

1. Introduction

Distributed machine learning has become common practice, given the increasing model complexity and the sheer size of real-world datasets. While GPUs have massively increased compute power, networks have not improved at the same pace. As a result, in large deployments with several parallel workers, distributed ML training is increasingly network bounded [9]. Parallelization techniques like mini-batch Stochastic Gradient Descent (SGD) alternate computation phases with model update exchanges among workers. In the widely-used synchronous setting, at the end of every SGD iteration, every node communicates hundreds of MBs of gradient values. This means that network performance often has a substantial impact on overall training time.

To prevent the network from becoming a bottleneck, several prior works [10, 1, 13, 12, 6, 3] have proposed lossy compression methods that reduce the communication data volume. These methods include sparsification methods that communicate only a fraction of the original gradients, and quantization methods that use fewer bits to represent the gradients. The main challenge underlying the design of compression methods is that the more compression is applied, the more information is lost, and the more will the compressed gradients differ from the original gradients, increasing its statistical variance. Higher variance implies slower convergence [1, 8], i.e., more communication rounds. So, compression offers a trade-off between the communication cost per iteration and the number of communication rounds.

In this work, we introduce a new, remarkably simple yet theoretically and practically effective compression technique, which we call *natural compression ($C_{\rm nat}$)*. Our technique is applied individually to all gradient values and works by randomized rounding to the nearest (negative or positive) power of two. $C_{\rm nat}$ is "natural" since the nearest power of two of a real value expressed as a float is computationally inexpensive and can be obtained by ignoring the mantissa. Thus, our scheme communicates just the exponents and signs of the original floats. Importantly, natural compression enjoys a provably small variance. The interested reader can find a complete theoretical analysis in our technical report [5].

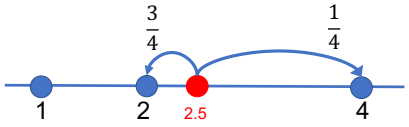

Figure 1: An illustration of natural compression applied to $t = 2.5$: $C_{\rm nat}(2.5) = 2$ with probability $\frac{4-2.5}{2} = 0.75$, and $C_{\rm nat}(2.5) = 4$ with probability $\frac{2.5-2}{2} = 0.25$.

Below, we briefly formalize $C_{\rm nat}$ and describe IntML, our prototype implementation of $C_{\rm nat}$ for TensorFlow. Through experimental results, we show that $C_{\rm nat}$ significantly reduces the training time compared to no compression. Our technical report includes more experiments as well as a comparison to other approaches.

2. Natural Compression

$C_{\rm nat}$ is a function mapping $t \in \mathbb{R}$ to a random variable $C_{\rm nat}(t) \in \mathbb{R}$. We define $C_{\rm nat}(0) = 0$. For $t > 0$, $C_{\rm nat}$ performs a stochastic rounding of t to either 2^a or 2^{a+1}, where a is such that $2^a \leq t < 2^{a+1}$. The rounding probabilities are chosen so as to ensure that the expected value of $C_{\rm nat}(t)$ is equal to t. For $t < 0$ the mapping is defined analogously. See Fig 1 for a graphical illustration.

Natural compression of a real number in a binary floating point format is computationally cheap. Regardless of the randomization step, $C_{\rm nat}$ amounts to simply dispensing off the mantissa in the binary representation. Thus, $C_{\rm nat}$ preserves 9 bits out of 32 bits, a $3.56\times$ improvement according to the IEEE 754 standard.

3. IntML Prototype Implementation

We implement the natural compression operator in C++ within the Gloo communication library [2], as a drop-in replacement for the ring all-reduce routine. We follow the same communication strategy introduced in SwitchML [9], which aggregates the gradients using In-Network Aggregation on programmable network switches. We prototype the In-Network Aggregation as a server-based program (we refer as aggregator) implemented atop DPDK [7], and leave a complete P4 implementation as future work; however, we

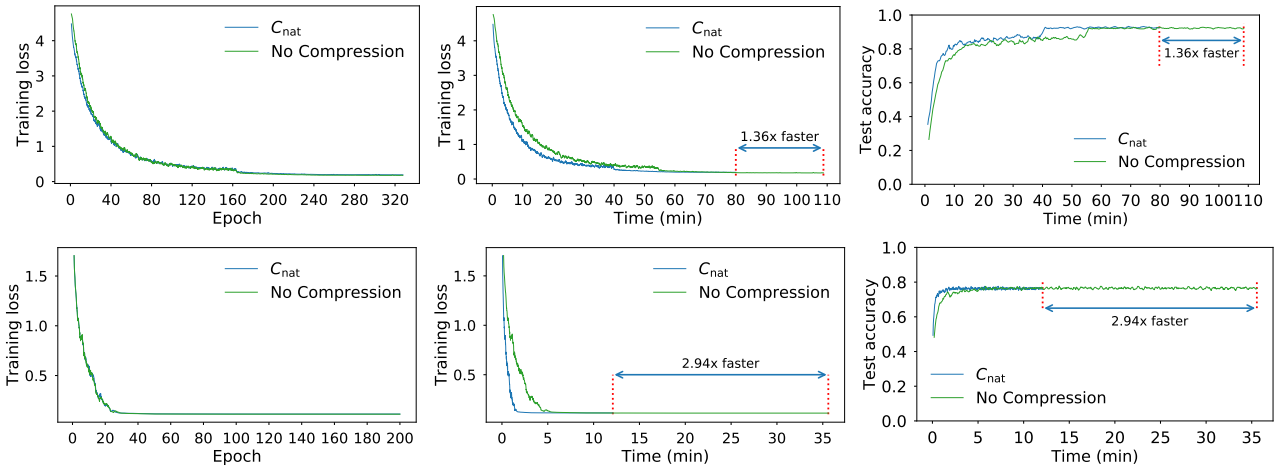

Figure 2: Training loss and test accuracy of ResNet110 (above) and AlexNet (below) on CIFAR10. Speedup is with respect to time to execute 320 and 200 epochs, respectively.

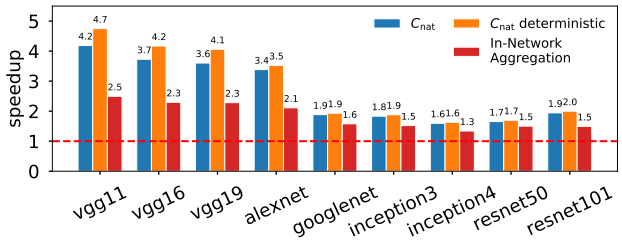

Figure 3: Training throughput speedup.

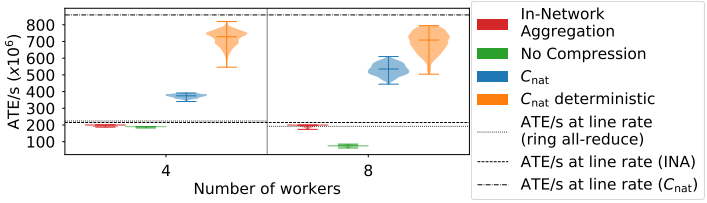

Figure 4: Violin plot of Aggregated Tensor Elements per second (ATE/s). Dashed lines denote the maximum ATE/s under line rate.

note that C_{nat} operations (bit shifting, masking, and random bits generation) are available on programmable switches.

We carefully optimize our implementation using modern x86 vector instructions (AVX2). To access memory more efficiently, we compress a 32-bit floating point number to an 8-bit representation, where 1 bit is for the sign of the number, 1 bit to indicate zero, and 6 bits are for the exponent. We clip the exponents in the range of $-50 \sim 10$. Note that, in our experiments, the exponent values never exceed the range of $-50 \sim 10$ and overflow or underflow never happened.

Despite the optimization effort, we identify non-negligible $10 \sim 15\%$ overheads in doing random number generation used in stochastic rounding, which was also reported in [6].

4. System Evaluation

We run the workers on 8 machines configured with 1 NVIDIA P100 GPU, dual CPU Intel Xeon E5-2630 v4 at 2.20 GHz, and 128 GB of RAM. We integrate the library with Horovod and, in turn, with TensorFlow. Our experiments execute the standard CNN benchmark [11].

We first elaborate the microbenchmark experiments of aggregated tensor elements (ATE) per second. Fig 4 shows the result for aggregating 200 tensors with the size of 100MB, where we vary the number of workers between 4 and 8. The performance difference observed for the case of C_{nat}, along with the similar performance for 4 and 8 workers for C_{nat} deterministic indicate that the overhead of doing stochastic rounding at the aggregator is a bottleneck.

We then illustrate the convergence behavior by training ResNet110 and AlexNet on CIFAR10 dataset. Fig 2 shows the training loss and test accuracy over time. We note that natural compression lowers training time by $\sim$26% for ResNet110 (17% more than QSGD for the same setup, see [1] (Table 1)) and 66% for AlexNet, compared to using no compression. The accuracy matches the results in [4] without loss of final accuracy with the same hyperparameters setting, training loss is also not affected by compression.

Next, we report the speedup measured in average training throughput while training benchmark CNN models on Imagenet dataset for one epoch. The throughput is calculated as the total number of images processed divided by the time elapsed. Fig 3 shows the speedup normalized by the training throughput of the baseline, that is, TensorFlow + Horovod using the NCCL communication library. We observe that the *communication-intensive* models (VGG, AlexNet) benefit more from quantization as compared to the *computation-intensive* models (GoogleNet, Inception, ResNet). These observations are consistent with prior work [1].

References

[1] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *NIPS*, 2017.

[2] Facebook. Gloo. https://github.com/facebookincubator/gloo.

[3] D. Grishchenko, F. Iutzeler, J. Malick, and M.-R. Amini. Asynchronous Distributed Learning with Sparse Communications and Identification. *CoRR*, abs/1812.03871, 2018. http://arxiv.org/abs/1812.03871.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] S. Horváth, C.-Y. Ho, v. Horváth, A. N. Sahu, M. Canini, and P. Richtárik. Natural Compression for Distributed Deep Learning. *CoRR*, abs/1905.10988, 2019. http://arxiv.org/abs/1905.10988.

[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*, 18(1):6869–6898, 2017.

[7] Intel. DPDK. https://www.dpdk.org/.

[8] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik. Distributed Learning with Compressed Gradient Differences. *CoRR*, abs/1901.09269, 2019. http://arxiv.org/abs/1901.09269.

[9] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik. Scaling Distributed Machine Learning with In-Network Aggregation. *CoRR*, abs/1903.06701, 2019. http://arxiv.org/abs/1903.06701.

[10] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. In *Interspeech*, 2014.

[11] TensorFlow benchmarks. https://github.com/tensorflow/benchmarks.

[12] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient Sparsification for Communication-Efficient Distributed Optimization. In *NIPS*, 2018.

[13] 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 *NIPS*, 2017.