Combining Global Sparse Gradients with Local Gradients in Distributed Neural Network Training

Alham Fikri Aji **Kenneth Heafield** **Nikolay Bogoychev**
School of Informatics, University of Edinburgh
10 Crichton Street
Edinburgh EH8 9AB
Scotland, European Union
{a.fikri, n.bogoych}@ed.ac.uk, kheafiel@inf.ed.ac.uk

Abstract

One way to reduce network traffic in multi-node data-parallel stochastic gradient descent is to only exchange the largest gradients. However, doing so damages the gradient and degrades the model's performance. Tranformer models degrade dramatically while the impact on RNNs is smaller. We restore gradient quality by combining the compressed global gradient with the node's locally computed uncompressed gradient. Neural machine translation experiments show that Transformer convergence is restored while RNNs converge faster. With our method, training on 4 nodes converges up to 1.5x as fast as with uncompressed gradients and scales 3.5x relative to single-node training.

1 Introduction

In recent years, neural network models have grown dramatically in terms of number of parameters (Wen et al., 2017), so exchanging gradients during data-parallel training is costly in terms of both bandwidth and time, especially in a distributed setting. Communication can be reduced (possibly at the expense of convergence) by sending only the top 1% of largest gradients in terms of absolute values, a method known as gradient dropping (Strom, 2015; Dryden et al., 2016; Aji and Heafield, 2017; Lin et al., 2018). Related methods are synchronizing less often (McMahan et al., 2017; Ott et al., 2018; Bogoychev et al., 2018) and quantization (Seide et al., 2014; Alistarh et al., 2016).

As these compression methods are lossy, each node's locally computed gradient is not immediately reflected in the global gradient. Our experiments show that gradient compression damages the model's performance, especially in the case of a Transformer model (Vaswani et al., 2017), which is known to be sensitive to noisy gradients (Chen et al., 2018; Ott et al., 2018; Aji and Heafield, 2019). We aim to repair the compressed gradient by combining it with the local gradients to improve the trade-off between convergence and compression rates.

In this paper, we apply gradient dropping to reduce the inter-node communication during distributed neural network training, which leads to faster training speed but reduced model convergence rate. We find that combining the sparse global gradient with the dense local gradient improves convergence. However, adding local information means that nodes' parameters will diverge over time. We address this by periodically averaging the model (McMahan et al., 2017), achieving faster end-to-end training time.

2 Related Work

2.1 Sparse Gradient Compression

Gradients are skewed: most values are near zero while very few have large absolute value (Aji and Heafield, 2017). Formally, Pearson's skewness coefficient is typically 2–4, but up to 262 in embedding matrices where much of the parameters lie. Sparse gradient compression exploits this by rounding gradients below a threshold to zero, sending only a sparse matrix of large gradients (Strom, 2015). The threshold can be set dynamically to the top 1% of gradients, achieving constant compression (Dryden et al., 2016). Unsent gradients are added to the next gradient prior to compression (Seide et al., 2014).

Gradient dropping is outlined in Algorithm 1. At each time step t, each node n computes a local gradient L_t^n on its data. The error feedback mechanism adds unsent gradients from the previous step E_{t-1} to the local gradient L_t^n. The combined gradient is then broken into sparse gradient S_t^n and residual E_t. Although the gradient is sparse, all

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 3624–3629,
Hong Kong, China, November 3–7, 2019.

Algorithm 1 Gradient dropping on node n

```
procedure SPARSESGD($L_t^n$)
    ▷ $L_t^n$ is local gradient of node $n$ at step $t$.
    $E_t^n \leftarrow E_{t-1}^n + L_t^n$
    $threshold \leftarrow K$-th largest of $|E_t^n|$
    $mask \leftarrow |E_t^n| \geq threshold$
    $S_t^n \leftarrow mask \odot E_t^n$
    $E_t^n \leftarrow \neg mask \odot E_t^n$
    $G_t \leftarrow AllReduce(S_t^n)$
    $ApplyOptimizer(G_t)$
end procedure
```

parameters are updated because Adam (Kingma and Ba, 2015) has momentum terms. Parameter updates run redundantly in all nodes so that only gradients are sent over the network.

The sum of sparse gradients is less sparse. We can send the summed gradient by itself (Lin et al., 2018) or again take the top 1% of summed gradients. Our cluster of 4 nodes is small enough that there was little speed difference, so we did not recompress the summed gradients.

2.2 Federated Averaging

Another way to reduce the bandwidth cost in multi-node training is by reducing the communication frequency (McMahan et al., 2017). In federated averaging, workers do not exchange gradients. Instead, each worker uses its local gradient to update its own local parameters. Each worker updates their local parameters by averaging across other nodes once every few steps.

In contrast with gradient dropping, federated averaging mainly uses the worker's local gradients for parameter updates. Gradients from other workers are not directly communicated and are therefore not taken into account by the optimizer.

3 Combining With Local Gradients

Recent work suggests that the Transformer is sensitive to noisy gradients, resulting in substantially worse models (Chen et al., 2018; Ott et al., 2018; Aji and Heafield, 2019). Consistent with these findings, both gradient sparsification and federated averaging yield low-quality Transformer models in our experiments. In gradient sparsification, noise comes from both thresholding and the error-feedback mechanism, which causes stale gradients. Federated averaging also introduces stale updates as this approach delays model synchronization. Previous work has shown that both noisy and stale gradients damage the model's quality (McMahan and Streeter, 2014; Ott et al., 2018; Dutta et al., 2018).

To address noisy updates in gradient sparsification, we combine the compressed global gradient and the uncompressed locally computed gradient in an effort to better approximate the true global gradient. Formally, let G_t be the compressed global gradient at time t and L_t^n be the gradient computed locally on node n. These will be combined into C_t^n that will be used to update the parameters.

An arguably naïve method sums the two gradients. With a scale-invariant optimizer like Adam, this is equivalent to averaging.

$$C_t^n = G_t + L_t^n$$

However, some of the locally-computed gradients were sent out and became part of the global gradient, so they will be double-counted by the sum. To compensate, we can subtract out the gradients S_t^n sent by node n.

$$C_t^n = G_t - S_t^n + L_t^n$$

The term $G_t - S_t^n$ equals to the sum of all sparse gradients from other nodes (or approximates it when the all-reduce compresses the result). The local gradient L_t^n used for updating does not include the error-feedback term E_t^n to prevent applying gradients multiple times while they are pending in error-feedback.

3.1 Periodic Synchronization

Nodes will diverge because local gradients differ. Therefore, models are averaged periodically. We average parameters (McMahan et al., 2017) every 500 steps with minimal impact on speed.

In the limit, a gradient is applied twice. First, a local update eventually makes its way to the other nodes via periodic averaging. Second, it accumulates with enough other gradients to be selected for inclusion in the compressed gradient and applied as part of a global update.

4 Experimental Setup

We use Marian (Junczys-Dowmunt et al., 2018) to train on nodes with 4xP100s. Multi-node experiments use 4 of these nodes, each connected with 40Gb Mellanox Infiniband. These scenarios will be abbreviated as 1x4 (one node with four GPUs) and 4x4 (four nodes with four GPUs each).

4.1 Model and Dataset

We perform our neural machine translation experiments on the following architectures.

Transformer: We train a Transformer model with six encoder and six decoder layers with tied embeddings. The model has 62M parameters. We train the model on the WMT 2017 English to German dataset with back-translated monolingual corpora (Sennrich et al., 2016b) and byte-pair encoding (Sennrich et al., 2016c), consisting of 19.1M sentence pairs. Model performance is validated on newstest2016 and tested on newstest2017.

Deep RNN: We also train a deep RNN model Sennrich et al. (2017) with eight layers of bidirectional LSTM consisting of 225M parameters. We train the model with the same English to German dataset from the Transformer experiment.

Shallow RNN: Our shallow RNN model is based on the winning system by Sennrich et al. (2016a) and is a single layer bidirectional encoder-decoder LSTM with attention consisting of 119M parameters. We train this model on WMT 2016 Romanian to English dataset, consisting of 2.5M sentence pairs. We also apply byte-pair encoding to this dataset. Model performance is validated on newsdev2016 and tested on newstest2016.

We apply layer normalization (Lei Ba et al., 2016) and exponential smoothing to train the model for 8 epochs of training.

4.2 Scaling Hyperparameters

In all our experiments, we use a memory budget of 10GB per GPU to dynamically fit as many sentences as possible, corresponding to an average of 450 and 250 sentences per batch per GPU for Ro-En and En-De, respectively. Hence, we apply several adjustments to the hyperparameters to accommodate the larger effective batch size of multi-node synchronous SGD:

Learning rate: The Adam optimizer is scale-invariant, so the parameters move at the same magnitude regardless the gradient size. Therefore, we linearly scale the learning rate in all multi-node experiments, as suggested by Goyal et al. (2017). On one node, we use a learning rate of 0.0003 for Transformer and deep RNN models, and 0.001 for the shallow RNN model. These values multiplied by 4 for the 4-node setting. The single-node learning rates were optimized in the sense that further increasing them damages performance.

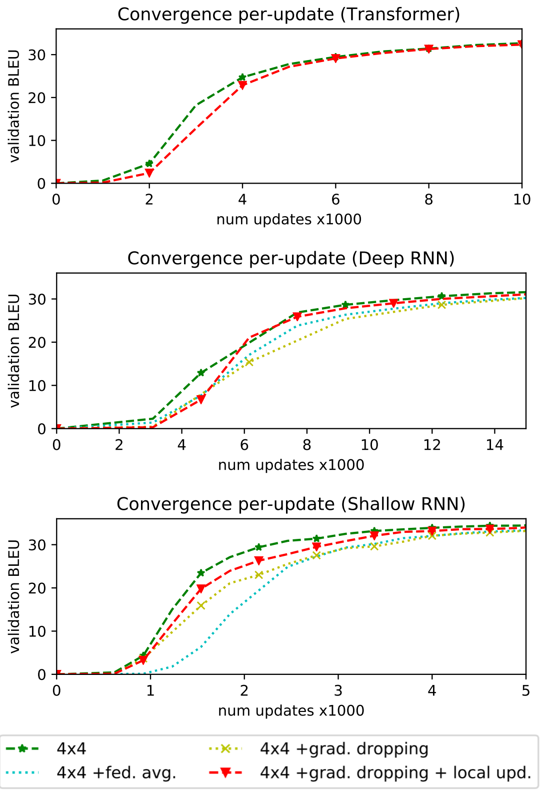

Figure 1: Model convergence per-updates on gradient dropping with local gradient update. We focus on the early stage of the training, before the BLEU converged. Training a Transformer with gradient dropping or federated averaging yielded 0 BLEU.

Warm-up: Learning rate warm-up helps overcome initial model instability when training with large mini-batches (Goyal et al., 2017). We add a linear learning rate warm-up for the Transformer, deep RNNs, and shallow RNNs for the first 16k, 4k, and 2k steps respectively. We apply inverse square root cool-down following Vaswani et al. (2017) for Transformer and deep RNN models.

We follow the rest of the hyperparameter settings as suggested in the papers (Vaswani et al., 2017; Sennrich et al., 2017, 2016a).

5 Results and Analysis

5.1 Restoring Quality

We approximate impact on quality by measuring the BLEU score (Papineni et al., 2002) obtained per update, experimenting with both RNN and Transformer systems. The baselines are vanilla 4-node synchronous SGD, gradient dropping (Aji and Heafield, 2017), and federated averaging (McMahan et al., 2017). For gradient dropping, we perform drop ratio warmup, gradually increasing it to 99% after 1000 steps. We average

	Transformer		Deep RNN		Shallow RNN	
Model	dev	test	dev	test	dev	test
Multi-node (4x4)	35.39	28.78	34.45	27.81	35.45	34.45
4x4 + gradient dropping	0.00	0.00	34.38	27.50	35.20	33.89
4x4 + federated averaging	0.00	0.00	34.33	27.42	35.25	33.93
4x4 + gradient dropping + local update	35.07	28.50	34.52	27.68	35.35	34.45

Table 1: Training quality of multi-node training with gradient compression techniques, measured with BLEU.

the model every 20 steps in federated averaging experiment, and every 500 steps in our proposed method.

Figure 1 shows BLEU score per update. Gradient dropping and federated averaging reduce gradient quality and improvement per update is slower. In the Transformer case, the model is incapable of training at all. Local gradient incorporation improves the sparse gradient quality and improves convergence per-epoch over gradient dropping. In all architectures, the model achieved a comparable training curve compared to the uncompressed multi-node training.

Table 1 summarizes model's performance in terms of BLEU. With local gradient incorporation, the models obtained better final quality, performing closer to uncompressed multi-node training. Local gradient incorporation enables the Transformer to train with a sparse gradient, albeit with slight quality degradation (0.28–0.32%). This result confirms Transformer's sensitivity to noisy updates and the ability of local gradients to mostly repair them.

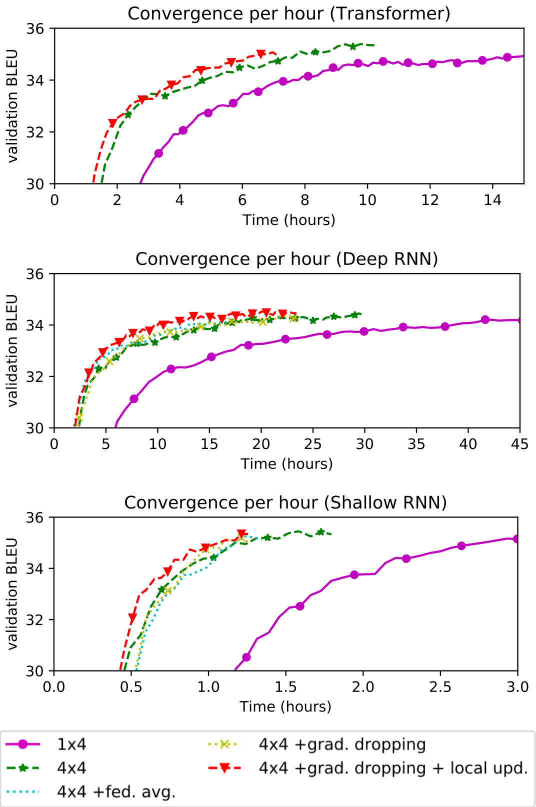

Figure 2: Convergence over time of gradient dropping with local gradient update.

5.2 Improving Training Speed

We measure the speed improvement of our proposed method by capturing the raw processing speed and time to reach certain BLEU. We compare it to both gradient dropping and federated averaging. We also measure the training efficiency by comparing the results with a single-node system. For the Transformer, we exclude gradient dropping and federated averaging as the models fail to train.

Table 2 summarizes our experiments. Gradient dropping reduces network traffic 50-fold and significantly improves the raw training speed in multi-node setting up to 3.4x over single-node, and up to 1.6x faster raw speed over uncompressed multi-node setting. Federated averaging is faster because there is no additional communication overhead for every step, and no extra computational cost for sparse gradients' compression. Finally, our method incurs the combined cost of gradient dropping, occasional federated averaging, and local updates so it is slower than gradient dropping at raw speed, but still substantially faster than uncompressed multi-node training.

While vanilla gradient dropping and federated averaging have better raw speed, there is no clear improvement on convergence speed as noisy gradients damage the convergence. Local gradient update restores the gradient and improves the convergence speed. In our RNN experiments, convergence speedup is closer to the raw speedup, up to 3.5x single-node performance.

The Transformer convergence rate increases more slowly than raw batch processing speed.

Model **Transformer (En-De)**	Words/ second	Raw speedup (1x4 / 4x4)	Time to conv.	conv. speedup (1x4 / 4x4)
Single-node (1x4)	36029	-	7.61h	-
Multi-node (4x4)	97111	2.7x / -	4.76h	1.6x / -
4x4 + gradient dropping + local update	122914	3.4x / 1.3x	4.03h	1.9x / 1.2x
Deep RNN (En-De)				
Single-node (1x4)	18205	-	23.68h	-
Multi-node (4x4)	42930	2.4x / -	10.59h	2.2x / -
4x4 + gradient dropping	60090	3.3x / 1.4x	8.94h	2.6x / 1.2x
4x4 + federated averaging	66149	3.6x / 1.5x	9.50h	2.5x / 1.1x
4x4 + gradient dropping + local update	59747	3.3x / 1.4x	6.80h	3.5x / 1.5x
Shallow RNN (Ro-En)				
Single-node (1x4)	36466	-	2.37h	-
Multi-node (4x4)	75641	2.1x / -	1.05h	2.3x / -
4x4 + gradient dropping	118189	3.2x / 1.6x	0.94h	2.5x / 1.1x
4x4 + federated averaging	124273	3.4x / 1.6x	1.06h	2.2x / 1.0x
4x4 + gradient dropping + local update	117756	3.2x / 1.6x	0.85h	2.8x / 1.2x

Table 2: Speed performance of gradient dropping with local gradient update, compared to several baselines. Time to convergence is time needed to reach 34.5 BLEU (Transformer & Shallow RNN) or 33.5 BLEU (Deep RNN).

While the rule of thumb is to scale learning rate linearly with batch size (Goyal et al., 2017), the Transformer model is also sensitive to high learning rates (Aji and Heafield, 2019). We obtained 1.6x convergence speedup, instead of the expected 2.7x speedup. Scaling the learning rate sublinearly can be explored.

Compression results are of course dependent on the ratio between computation and network bandwidth in a system, as well as model size. Because the method reduces network load, we would expect to see even larger speed improvement with commodity hardware instead of the 40 gigabit Infiniband network used in our experiments.

6 Conclusion

We improve model convergence when training with sparse gradients by utilizing an additional locally-computed gradient, while also negates the quality loss in terms of BLEU caused by gradient dropping. With gradient dropping and local gradient incorporation, we improve the raw training speed in terms of word/second by up to 3.4x over single-node system, and up to 1.6x over uncompressed multi-node system. We also evaluate the training speed by the time needed to reach a near-convergence BLEU score. In this case, we improve the training speed by up to 3.5x over single-node system and up to 1.5x over uncompressed multi-node system.

7 Acknowledgements

Alham Fikri Aji is funded by the Indonesia Endowment Fund for Education scholarship scheme. This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (`http://www.csd3.cam.ac.uk/`), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council (`www.dirac.ac.uk`).

References

Alham Fikri Aji and Kenneth Heafield. 2017. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 440–445.

Alham Fikri Aji and Kenneth Heafield. 2019. Making asynchronous stochastic gradient descent work for transformers. *arXiv preprint arXiv:1906.03496*.

Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: randomized quantization for communication-optimal stochastic gradient descent. *CoRR*, abs/1610.02132.

Nikolay Bogoychev, Kenneth Heafield, Alham Fikri Aji, and Marcin Junczys-Dowmunt. 2018. Accelerating asynchronous stochastic gradient descent for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2991–2996.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, et al. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86.

Nikoli Dryden, Sam Ade Jacobs, Tim Moon, and Brian Van Essen. 2016. Communication quantization for data-parallel training of deep neural networks. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, pages 1–8. IEEE Press.

Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. 2018. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International Conference on Artificial Intelligence and Statistics*, pages 803–812.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*.

J. Lei Ba, J. R. Kiros, and G. E. Hinton. 2016. Layer Normalization. *ArXiv e-prints*.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. 2018. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282.

Brendan McMahan and Matthew Streeter. 2014. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems*, pages 2915–2923.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evalution of machine translation. In *Proceedings 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, PA.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*.

Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. 2017. The University of Edinburgh's neural mt systems for WMT17. In *Proceedings of the Second Conference on Machine Translation*, pages 389–399.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Edinburgh neural machine translation systems for WMT 16. In *Proceedings of the ACL 2016 First Conference on Machine Translation (WMT16)*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016c. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725.

Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *INTERSPEECH*, volume 7, page 10.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pages 1509–1519.