COMBINING GLOBAL SPARSE GRADIENTS WITH LOCAL GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data-parallel neural network training is network-intensive, so gradient dropping was designed to exchange only large gradients. However, gradient dropping has been shown to slow convergence. We propose to improve convergence by having each node combine its locally computed gradient with the sparse global gradient exchanged over the network. We empirically confirm with machine translation tasks that gradient dropping with local gradients approaches convergence 48% faster than non-compressed multi-node training and 28% faster compared to vanilla gradient dropping. We also show that gradient dropping with a local gradient update does not reduce the model's final quality.

1 INTRODUCTION

Training a neural network can be slow, especially with a large model or dataset (Raina et al., 2009; Wen et al., 2017). Distributed training is becoming essential to speed up the process. In data-parallel training, multiple workers optimize the same parameters based on different parts of the training data then exchange parameters.

Data-parallel training is network intensive because workers send and fetch gradients that have the same size as the model. Several techniques have been proposed to reduce the traffic in data-parallelism training by using quantization to compress the gradient sent (Seide et al., 2014; Alistarh et al., 2016) or selecting sparse matrices (Strom, 2015; Dryden et al., 2016; Aji & Heafield, 2017; Lin et al., 2017).

Gradient dropping, and its extension Deep Gradient Compression (Lin et al., 2017), is a recent approach that compresses the network by sending a small fraction (about 1%) of the largest gradients (by absolute value). This technique is based on the observation that the gradient values are skewed, as most are close to zero. An issue with gradient compression is that gradients are compressed so much that it slows the model's convergence rate and can reduce the model's final quality (Aji & Heafield, 2017).

In vanilla gradient dropping, all nodes update with the same sparse gradient exchanged over the network, while other parameters are unchanged. However, each node has computed a local gradient on its own data. Can we exploit this dense local gradient alongside the sparse global gradient to improve convergence? We propose and evaluate three ways to combine them.

2 RELATED WORK ON GRADIENT COMPRESSION

Compression is a natural way to reduce network traffic. 1-Bit SGD (Seide et al., 2014) and QSGD (Alistarh et al., 2016) quantize the gradient into a 1-bit matrix. Strom (2015) proposed a threshold-based quantization where only gradients larger than the threshold are sent. Dryden et al. (2016) and Aji & Heafield (2017) adapt this idea to instead send the top X% of the gradients. Alternatively, gradients from several mini-batches can be accumulated before applying an update to reduce the network cost (Dean et al., 2012; Ott et al., 2018; Bogoychev et al., 2018). Gradient compression can be seen as a continuous version of gradient accumulation: instead of exchanging all gradients every so often, large gradients are exchanged rapidly while small gradients accumulate locally.

Algorithm 1 Gradient dropping on node n

1: **procedure** SPARSESGD(L_t^n)
2: ▷ L_t^n is a local gradient computed by node n at step t.
3: $E_t^n \leftarrow E_{t-1}^n + L_t^n$
4: $threshold \leftarrow K$-th largest of $|E_t^n|$
5: $mask \leftarrow |E_t^n| \geq threshold$
6: $S_t^n \leftarrow mask \odot E_t^n$
7: $E_t^n \leftarrow \neg mask \odot E_t^n$
8: $G_t \leftarrow AllReduce(S_t^n)$
9: $ApplyOptimizer(G_t)$
10: **end procedure**

2.1 GRADIENT DROPPING

Gradient dropping compresses communication by selecting the top 1% of gradients from each node by absolute value. An optimizer, such as SGD or Adam (Kingma & Ba, 2015), uses the summed sparse gradients from all nodes. An error feedback mechanism stores unsent gradients and adds them to the next gradient update before compression (Seide et al., 2014). Without this mechanism, the model will not converge.

Formally, gradient dropping is outlined in Algorithm 1. For each time step t, each node n produces a local gradient L_t^n. First, we apply error feedback by adding the unsent gradients from past 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. We combine sparse gradients from every node by using all-reduce and use it to update the parameter. Although the gradient is sparse, we apply a parameter update on the entire parameter. This is done to let the optimizer update its momentum and apply momentum movement to the entire parameter.

Gradient dropping significantly reduces the communication cost in data-parallel training, making each parameter update faster but degrading convergence per update. When network bandwidth is the bottleneck, overall time to convergence is reduced.

2.2 DEEP GRADIENT COMPRESSION

Deep gradient compression (DGC) (Lin et al., 2017) introduces four modifications that reduces the quality damage of gradient dropping. For momentum-based optimizers like Adam (Kingma & Ba, 2015), they apply a correction for missing gradients and masking in the error feedback mechanism. Additionally, DGC applies gradient clipping locally before compression instead of globally before applying an optimizer. DGC also warms up the compression rate, sending more in early training. Our work differs in that we use the local gradients so every parameter has an update. Masking therefore does not directly port to our work, while gradient clipping and compression warm-up are equally applicable.

3 COMBINING WITH LOCAL GRADIENT

The issue with gradient dropping is that through the use of lossy compression to compress the gradient, damage is caused in the gradient making the model harder to learn. DGC fixes this issue to some extent, but the model still relies on a sparse gradient. We may as well also use the dense locally computed gradient if we can find a way to combine the two.

Formally, we currently only use the compressed global gradient G_t, as in Algorithm 1 line 9, to update the model. Instead, we incorporate the local gradient context to gain a better approximation of the compressed gradient. Let L_t^n be gradient computed locally at time t on node n. Our goal is to compute a combined gradient C_t^n incorporated with local gradient context from node n at time t. As described in Algorithm 2, we propose three formulas to obtain C_t^n that will be used to update the parameter.

Algorithm 2 Gradient dropping with local gradient incorporation on node n

procedure SPARSESGD(L_t^n)
  ▷ L_t^n is a local gradient computed by node n at step t.
  $E_t^n \leftarrow E_{t-1}^n + L_t^n$
  $threshold \leftarrow K$-th largest of $|E_t^n|$
  $mask \leftarrow |E_t^n| \geq threshold$
  $S_t^n \leftarrow mask \odot E_t^n$
  $E_t^n \leftarrow \neg mask \odot E_t^n$
  $G_t \leftarrow AllReduce(S_t^n)$
  switch $mode$ **do**
    case SUM
      $C_t^n = (G_t + L_t^n)/2$
    case PARTIAL
      $C_t^n = G_t - S_t^n + L_t^n$
    case ERROR
      $C_t^n = G_t + E_t^n$
      $E_t^n = 0$
  $ApplyOptimizer(C_t^n)$
end procedure

SUM We use the local gradient from each node to predict the general direction of the global gradient. An arguably naïve way to incorporate the local gradient is to add it to the sparse global gradient by

$$C_t^n = (G_t + L_t^n)/2$$

where we divide the sum by 2 to avoid double counting the computed gradient. We can ignore this if we apply a scale-invariant optimizer such as Adam.

PARTIAL Since some of the local gradients L_t^n make their way into the sparse global gradient G_t, it seems unfair to count them twice in the SUM method. We can correct for this by subtracting the locally-generated sparse gradients S_t^n.

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

where the term $G_t - S_t^n$ is equal to the sum of the sparse gradients from every node except the local node. Therefore, we only use sparse gradients from the other nodes while using a non-sparsified local gradient.

ERROR Finally, we attempt to incorporate the residual error stored in E_t^n by simply adding them to the sparse gradient. However, to avoid using the same gradient over and over, we have to clear the residual. Therefore, in this approach, instead of accumulating the unsent gradients, we just apply them as a local context instead. We update the parameter with

$$C_t^n = G_t + E_t^n$$

$$E_t^n = 0$$

Clearing the error at every step is equivalent to removing the error feedback mechanism. As the error E_t^n is now only contains the current step's unsent gradient which is equal to $L_t^n - S_t^n$, ERROR is equivalent to PARTIAL without an error-feedback mechanism.

Parameters on each node will diverge because each local gradient is different. To resolve this, we periodically average the parameters across nodes. Since we must communicate all parameters, the synchronization should be infrequent enough that it does not significantly effect the overall training speed in terms of words per second. In this research, we are also considering averaging the parameters across all nodes for every 500 steps.

4 EXPERIMENTAL SETUP

Using Marian (Junczys-Dowmunt et al., 2018) as the toolkit, we train on nodes with four P100s each. Each node is connected with 40Gb Infiniband from Mellanox, and our multi-node experiments use four nodes.

4.1 MODEL AND DATASET

We test our experiment on the following tasks.

Ro-En Machine Translation: We build a Romanian-to-English neural machine translation system using all the parallel corpora in the constrained WMT 2016 task (Bojar et al., 2016). The dataset consists of 2.6M pairs of sentences to which we apply byte-pair encoding (Sennrich et al., 2016a). Our model is based on the winning system by Sennrich et al. (2016b) and is a single layer attentional encoder-decoder bidirectional LSTM consisting of 119M parameters. We apply layer normalization (Lei Ba et al., 2016) and exponential smoothing to train the model for up to 14 epochs or until no improvement after five validations. We optimize the model with the Adam optimizer.

Training the Ro-En NMT system with this model is fast, so we primarily use the dataset and model for our development experiments.

En-De Machine Translation: We train another machine translation system on English-to-German for our large and high-resource model. The corpus consists of 19.1M pairs of sentences after back-translation. The model is based on the winning system by Sennrich et al. (2017) with eight layers of LSTM consisting of 225M parameters. The configuration is similar to the previous system as we use layer normalization and exponential smoothing, train the model for up to eight epochs and optimize with Adam. Concerned with time to convergence of a single model, we report single model scores rather than ensemble scores.

4.2 HYPERPARAMETERS

The baseline systems were trained on a single node (Sennrich et al., 2017) but our experiments focus on multi-node settings. Thus, we apply several adjustments to the hyperparameters to accommodate the larger effective batch size of synchronous stochastic gradient descent. These hyperparameters are used in multi-node baselines and experimental conditions.

Synchronous: We follow Lin et al. (2017) in applying synchronous stochastic gradient descent, so that nodes can easily aggregate internally amongst their GPUs and exchange one copy of the gradients externally. This differs from Aji & Heafield (2017) where experiments were asynchronous but confined within a node.

Batch size: Prior work (Aji & Heafield, 2017; Lin et al., 2017) on gradient dropping used relatively small batch sizes even though larger batches are generally faster due to using more of the GPU. In all our experiments, we use a workspace of 10GB and dynamically fit as many sentences as possible for each batch, which provides an average of 450 and 250 sentences per batch per GPU for Ro-En and En-De, respectively.

Learning rate: The Adam optimizer is scale-invariant, so the parameter moves at the same magnitude with both single and multi-node settings despite having approximately 4x larger gradients in the multi-node. Therefore, we linearly scale the learning rate by 4x in all experiments to resolve this, as suggested by Goyal et al. (2017). We use a learning rate of 0.002 in the Ro-En multi-node and 0.0005 in the Ro-En single-node. Similarly, we use a learning rate of 0.0012 in the En-De multi-node and 0.0003 in the En-De single-node.

Warm-up: Learning rate warm-up also helps in training with large mini-batch scenario to overcome model instability during the early stages of training (Goyal et al., 2017). So, we add a learning rate warm-up for all multi-node experiments by linearly increasing the rate until it reaches the desired amount after several steps. We apply a warm-up for the first 2000 steps in the Ro-En and 4000 steps in the En-De experiments. To provide a fair comparison, we also apply the warm-up for the same number of examples in the multi-node and single-node experiments.

The remaining hyperparameters are equivalent in both single-node and multi-node settings.

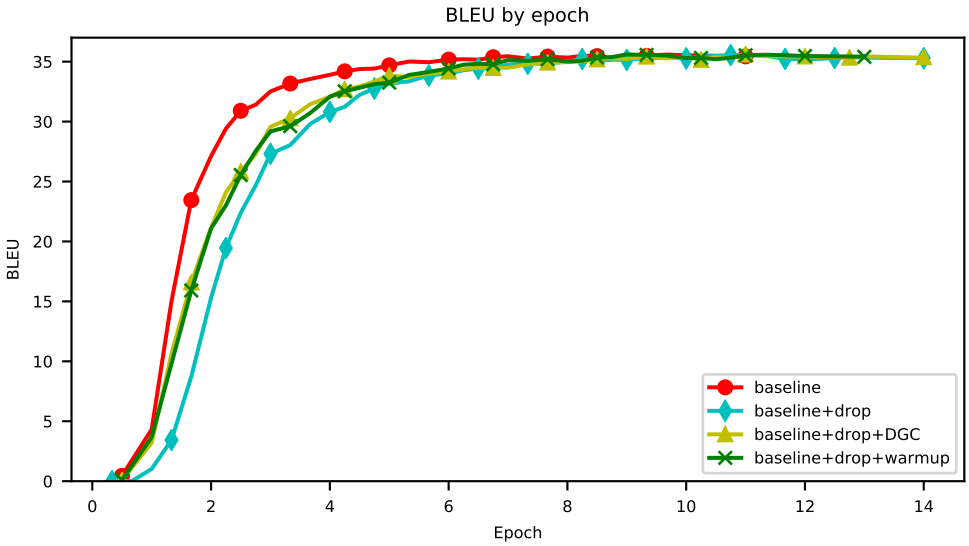

Figure 1: Convergence by epoch on Romanian–English training. Gradient dropping takes more epochs (but not necessarily time) to reach peak performance even with deep gradient compression and ratio warm-up.

5 Results and Analysis

5.1 Drop Ratio Warm-up

Gradient dropping increases the raw training speed from 73k words/second to 116k words/second in the multi-node Ro-En experiment. However, gradient dropping also damages convergence in the sense that it takes more epochs to reach peak performance.

DGC is proposed to minimize the convergence damage caused by gradient dropping. While DGC typically performs better than gradient dropping, we argue that most of the improvement is due to the compression ratio warm-up. To confirm this, we ran an experiment on the multi-node Ro-En with a drop ratio warm-up. At the tth step, we discard R_t of the gradient, defined below, with a warm-up period T. In this case we set $T = 1000$, which is equal to 3 epochs in Ro-En experiment.

$$R_t = 0.99^{max(1, \frac{T}{t})}$$

The result shown in Figure 1 suggests that the compression ratio warm-up can improve the convergence with gradient dropping. On the other hand, there is no contrast in terms of convergence by other methods proposed in DGC. Based on this result, we choose to use compression ratio warm-up for the remainder of the experiments.

5.2 Local Gradient Update

We test our proposed techniques to incorporate the local gradient while performing a sparse gradient update. We base the experiment in a multi-node setting with gradient dropping configured with a dropping ratio of 99% and a dropping rate warm-up for the first three epochs on the Ro-En dataset. We also apply each of our local gradient update techniques.

Figure 2 shows convergence after incorporating the local gradient. Using the Partial or Sum update techniques improves the convergence curve in the early stages of the training as the convergence curves are closer to the baseline. However, the curves are becoming unstable after several epochs. Finally, we can see that their final qualities are lower, which we attribute to divergence of the models. However, it is interesting that the models are still capable of learning even with model inconsistencies between workers.

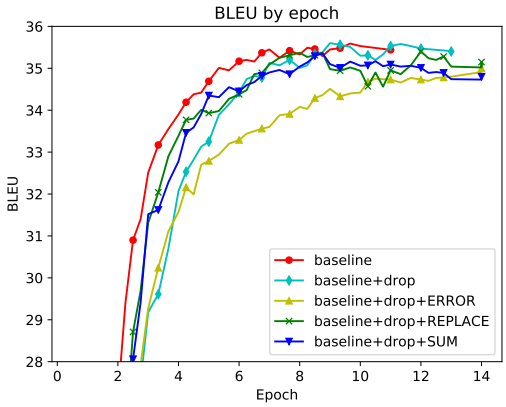

Figure 2: Convergence per step of gradient dropping with various local gradient update techniques on the Ro-En multi-node experiment.

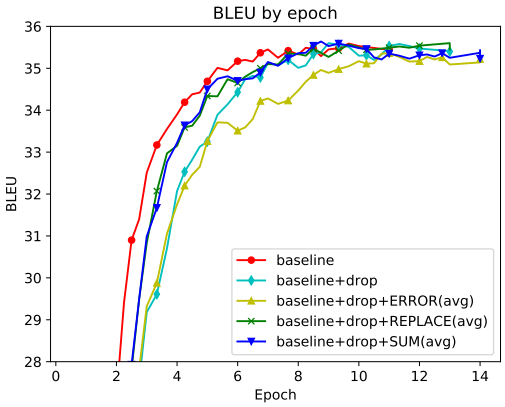

Figure 3: Convergence per step of gradient dropping with various local gradient update techniques and periodic model synchronization on the Ro-En multi-node experiment.

Table 1: Performance of local gradient update on the Ro-En multi-node experiment.

Model	time to reach 34.5 BLEU	training time	CE	Dev. BLEU	Test BLEU
baseline	1.38h	3.00h	50.6	35.59	34.45
baseline+drop	1.28h	2.81h	51.4	35.6	33.89
baseline+drop+ERROR	2.08h	2.91h	53.0	35.05	33.33
baseline+drop+ERROR+sync	1.64h	2.91h	52.2	35.36	33.8
baseline+drop+PARTIAL	1.33h	2.91h	51.1	35.4	34.34
baseline+drop+PARTIAL+sync	1.19h	**2.58h**	50.8	35.6	34.45
baseline+drop+SUM	1.31h	2.97h	51.1	35.36	34.15
baseline+drop+SUM+sync	**1.07h**	2.91h	51.1	35.64	34.34

We apply periodic model synchronization by doing model averaging across all workers at every 500 steps. As shown in Figure 3, the model is capable of learning and maintaining the same final quality as the baseline.

To understand the performance better, we capture several details provided in Table 1. Without synchronization, the model suffers a reduced quality in the development and test BLEU scores. Using the ERROR local gradient update technique does not seem to benefit the model. On the other hand, using PARTIAL or SUM with periodic synchronization significantly improves the convergence curve of the gradient dropping technique, and using PARTIAL appears to provide a more stable result compared to SUM. PARTIAL also helps the model obtain better cross-entropy and reach convergence faster, thereby reducing the training time.

5.3 END-TO-END EXPERIMENTS

We train a model on 4 nodes with 4 GPUs each (henceforth 4x4) with gradient dropping, drop ratio warm-up for 1000 steps, local gradient update using the PARTIAL strategy, and periodic synchronization. The baselines are a single-node configuration with 4 GPUs (denoted 1x4) and a 4x4 multi-node configuration, both with ordinary synchronous SGD. Additionally, we try a 4x4 configuration with gradient dropping with drop ratio warmup for 1000 steps.

Table 2 summarizes our end-to-end experimental results. In the ideal case, using 4x more workers should provide a 4x speed improvement. Compressing the gradient reduces the network cost and significantly improves the raw words/second speed by about 3x over a single-node experiment. Using local gradient update slightly decreases the average speed as it requires extra communication cost for the periodic synchronization.

Table 2: Performance of gradient dropping with local gradient update on different tasks.

Ro-En

Model	words/ second	time to reach 34.5 BLEU	training time	CE	Dev. BLEU	Test. BLEU
Single-node (1x4)	36058	2.55h	6.39h	50.1	35.56	34.4
Multi-node (4x4)	73216	1.38h	3.00h	50.6	35.59	34.45
+ gradient dropping	116424	1.28h	2.81h	51.4	35.6	33.89
+ local gradient update	112060	1.19h	2.58h	50.8	35.6	34.45

En-De

Model	words/ second	time to reach 33.5 BLEU	training time	CE	Dev. BLEU	Test. BLEU
Single-node (1x4)	19007	24.94h	63.71h	37.4	34.38	27.48
Multi-node (4x4)	41020	11.69h	31.67h	35.7	34.45	27.81
+ gradient dropping	58000	10.12h	23.58h	36.1	34.38	27.50
+ local gradient update	56836	7.89h	24.37h	35.8	34.52	27.68

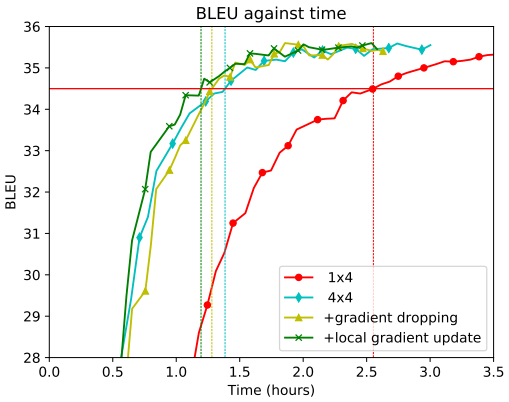

Figure 4: Convergence over time of gradient dropping with local gradient update on the Ro-En task.

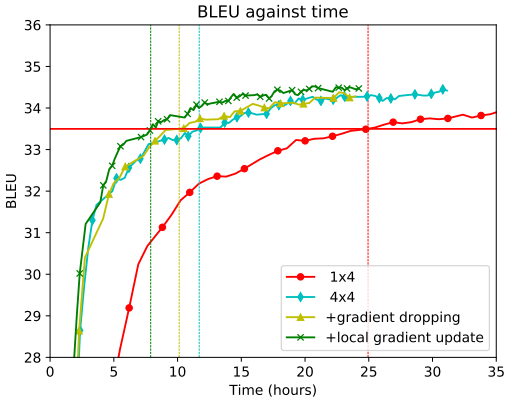

Figure 5: Convergence over time of gradient dropping with local gradient update on the En-De task.

Although slightly slower, local gradient update significantly improves the convergence speed as shown in Figures 4 and 5. In both cases, vanilla gradient dropping massively increases the raw speed, there is no clear improvement on overall convergence time compared to the uncompressed multi-node training. We significantly reduce training time and the time to reach a near-convergence BLEU by using a local gradient update. It also shows that local gradient update reduces the quality damage caused by gradient dropping. Note that the improvement in words/second is greater compared to the training time in the En-De experiment because the model spends additional time for data and I/O operations (e.g., model saving and loading or data shuffling and reading).

6 Conclusion

We significantly reduce convergence damage caused by compressing the gradient through gradient dropping in data-parallelism training. We utilize a locally-computed gradient to predict and reconstruct the dense gradient. Our experiments show that we can improve the training time up to 45% faster compared to a non-compressed multi-node system and 3x faster compared to a single-node system. Local gradient update is also empirically shown to negate the quality loss caused by gradient dropping.

REFERENCES

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

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

Nikolay Bogoychev, Marcin Junczys-Dowmunt, Kenneth Heafield, and Alham Fikri Aji. Accelerating asynchronous stochastic gradient descent for neural machine translation. *arXiv preprint arXiv:1808.08859*, 2018.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation*, pp. 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W16/W16-2301.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Nikoli Dryden, Sam Ade Jacobs, Tim Moon, and Brian 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*, pp. 1–8. IEEE Press, 2016.

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

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. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pp. 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P18-4020.

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

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

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880. ACM, 2009.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 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*, 2014.

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

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

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

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

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