Scaling Neural Machine Translation

Myle Ott[△] **Sergey Edunov**[△] **David Grangier**[▽*] **Michael Auli**[△]
[△]Facebook AI Research, Menlo Park & New York.
[▽]Google Brain, Mountain View.

Abstract

Sequence to sequence learning models still require several days to reach state of the art performance on large benchmark datasets using a single machine. This paper shows that reduced precision and large batch training can speedup training by nearly 5x on a single 8-GPU machine with careful tuning and implementation.[1] On WMT'14 English-German translation, we match the accuracy of Vaswani et al. (2017) in under 5 hours when training on 8 GPUs and we obtain a new state of the art of 29.3 BLEU after training for 85 minutes on 128 GPUs. We further improve these results to 29.8 BLEU by training on the much larger Paracrawl dataset. On the WMT'14 English-French task, we obtain a state-of-the-art BLEU of 43.2 in 8.5 hours on 128 GPUs.

1 Introduction

Neural Machine Translation (NMT) has seen impressive progress in the recent years with the introduction of ever more efficient architectures (Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017). Similar sequence-to-sequence models are also applied to other natural language processing tasks, such as abstractive summarization (See et al., 2017; Paulus et al., 2018) and dialog (Sordoni et al., 2015; Serban et al., 2017; Dusek and Jurcícek, 2016).

Currently, training state-of-the-art models on large datasets is computationally intensive and can require several days on a machine with 8 high-end graphics processing units (GPUs). Scaling training to multiple machines enables faster experimental turn-around but also introduces new challenges: How do we maintain efficiency in a distributed setup when some batches process faster than others (i.e., in the presence of *stragglers*)? How do larger batch sizes affect optimization and generalization performance? While stragglers primarily affect multi-machine training, questions about the effectiveness of large batch training are relevant even for users of commodity hardware on a single machine, especially as such hardware continues to improve, enabling bigger models and batch sizes.

In this paper, we first explore approaches to improve training efficiency on a single machine. By training with reduced floating point precision we decrease training time by 65% with no effect on accuracy. Next, we assess the effect of dramatically increasing the batch size from 25k to over 400k tokens, a necessary condition for large scale parallelization with synchronous training. We implement this on a single machine by accumulating gradients from several batches before each update. We find that by training with large batches and by increasing the learning rate we can further reduce training time by 40% on a single machine. Finally, we parallelize training across 16 machines and find that we can reduce training time by an additional 90% compared to a single machine.

Our improvements enable training a Transformer model on the WMT'16 En-De dataset to the same accuracy as Vaswani et al. (2017) in just 32 minutes on 128 GPUs and in under 5 hours on 8 GPUs. This same model trained to full convergence achieves a new state of the art of 29.3 BLEU in 85 minutes. These scalability improvements additionally enable us to train models on much larger datasets. We show that we can reach 29.8 BLEU on the same test set in less than 10 hours when trained on a combined corpus of WMT and Paracrawl data containing ∼150M sentence pairs (i.e., over 30x more training data). Similarly, on the WMT'14 En-Fr task we obtain a state of the art BLEU of 43.2 in 8.5 hours on 128 GPUs.

*Work done while at Facebook AI Research.

[1]Our implementation is available at: https://www.github.com/pytorch/fairseq

Proceedings of the Third Conference on Machine Translation (WMT), Volume 1: Research Papers, pages 1–9

https://doi.org/10.18653/v1/W18-64001

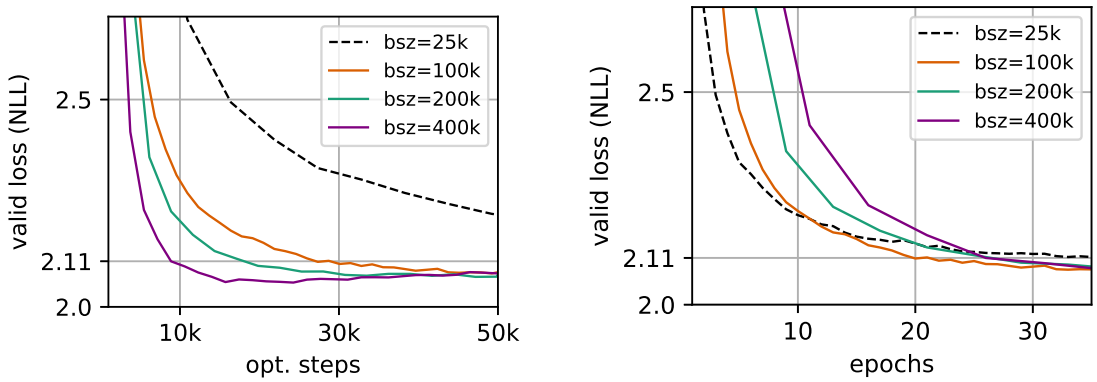

Figure 1: Validation loss for Transformer model trained with varying batch sizes (bsz) as a function of optimization steps (left) and epochs (right). Training with large batches is less data-efficient, but can be parallelized. Batch sizes given in number of target tokens excluding padding. *WMT En-De, newstest13*.

2 Related Work

Previous research considered training and inference with reduced numerical precision for neural networks (Simard and Graf, 1993; Courbariaux et al., 2015; Sa et al., 2018). Our work relies on half-precision floating point computation, following the guidelines of Micikevicius et al. (2018) to adjust the scale of the loss to avoid underflow or overflow errors in gradient computations.

Distributed training of neural networks follows two main strategies: (i) *model parallel* evaluates different model layers on different workers (Coates et al., 2013) and (ii) *data parallel* keeps a copy of the model on each worker but distributes different batches to different machines (Dean et al., 2012). We rely on the second scheme and follow synchronous SGD, which has recently been deemed more efficient than asynchronous SGD (Chen et al., 2016). Synchronous SGD distributes the computation of gradients over multiple machines and then performs a synchronized update of the model weights. Large neural machine translation systems have been recently trained with this algorithm with success (Dean, 2017; Chen et al., 2018).

Recent work by Puri et al. (2018) considers large-scale distributed training of language models (LM) achieving 109x scaling with 128 GPUs. Compared to NMT training, however, LM training does not face the same challenges of variable batch sizes. Moreover, we find that large batch training requires warming up the learning rate, whereas their work begins training with a large learning rate. There has also been recent work on using lower precision for inference only (Quinn and Ballesteros, 2018).

Another line of work explores strategies for improving communication efficiency in distributed synchronous training setting by abandoning "stragglers," in particular by introducing redundancy in how the data is distributed across workers (Tandon et al., 2017; Ye and Abbe, 2018). The idea rests on coding schemes that introduce this redundancy and enable for some workers to simply not return an answer. In contrast, we do not discard any computation done by workers.

3 Experimental Setup

3.1 Datasets and Evaluation

We run experiments on two language pairs, English to German (En–De) and English to French (En–Fr). For En–De we replicate the setup of Vaswani et al. (2017) which relies on the WMT'16 training data with 4.5M sentence pairs; we validate on newstest13 and test on newstest14. We use a vocabulary of 32K symbols based on a joint source and target byte pair encoding (BPE; Sennrich et al. 2016). For En–Fr, we train on WMT'14 and borrow the setup of Gehring et al. (2017) with 36M training sentence pairs. We use newstest12+13 for validation and newstest14 for test. The 40K vocabulary is based on a joint source and target BPE factorization.

We also experiment with scaling training beyond 36M sentence pairs by using data from the Paracrawl corpus (ParaCrawl, 2018). This dataset is extremely large with more than 4.5B pairs for En–De and more than 4.2B pairs for

En–Fr. We rely on the BPE vocabulary built on WMT data for each language pair and explore filtering this noisy dataset in Section 4.5. We measure case-sensitive tokenized BLEU with `multi-bleu.pl`[2] and de-tokenized BLEU with SacreBLEU[3] (Post, 2018). All results use beam search with a beam width of 4 and length penalty of 0.6, following Vaswani et al. 2017. Checkpoint averaging is not used, except where specified otherwise.

3.2 Models and Hyperparameters

We use the Transformer model (Vaswani et al., 2017) implemented in PyTorch in the `fairseq-py` toolkit (Edunov et al., 2017). All experiments are based on the "big" transformer model with 6 blocks in the encoder and decoder networks. Each encoder block contains a self-attention layer, followed by two fully connected feed-forward layers with a ReLU non-linearity between them. Each decoder block contains self-attention, followed by encoder-decoder attention, followed by two fully connected feed-forward layers with a ReLU between them. We include residual connections (He et al., 2015) after each attention layer and after the combined feed-forward layers, and apply layer normalization (Ba et al., 2016) after each residual connection. We use word representations of size 1024, feed-forward layers with inner dimension 4,096, and multi-headed attention with 16 attention heads. We apply dropout (Srivastava et al., 2014) with probability 0.3 for En-De and 0.1 for En-Fr. In total this model has 210M parameters for the En-De dataset and 222M parameters for the En-Fr dataset.

Models are optimized with Adam (Kingma and Ba, 2015) using $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 1\text{e}{-8}$. We use the same learning rate schedule as Vaswani et al. (2017), i.e., the learning rate increases linearly for 4,000 steps to $5\text{e}{-4}$ (or $1\text{e}{-3}$ in experiments that specify `2x lr`), after which it is decayed proportionally to the inverse square root of the number of steps. We use label smoothing with 0.1 weight for the uniform prior distribution over the vocabulary (Szegedy et al., 2015; Pereyra et al., 2017).

All experiments are run on DGX-1 nodes with 8 NVIDIA© V100 GPUs interconnected by Infiniband. We use the NCCL2 library and `torch.distributed` for inter-GPU communication.

4 Experiments and Results

In this section we present results for improving training efficiency via reduced precision floating point (Section 4.1), training with larger batches (Section 4.2), and training with multiple nodes in a distributed setting (Section 4.3).

4.1 Half-Precision Training

NVIDIA Volta GPUs introduce Tensor Cores that enable efficient half precision floating point (FP) computations that are several times faster than full precision operations. However, half precision drastically reduces the range of floating point values that can be represented which can lead to numerical underflows and overflows (Micikevicius et al., 2018). This can be mitigated by scaling values to fit into the FP16 range.

In particular, we perform all forward-backward computations as well as the all-reduce (gradient synchronization) between workers in FP16. In contrast, the model weights are also available in full precision, and we compute the loss and optimization (e.g., momentum, weight updates) in FP32 as well. We scale the loss right after the forward pass to fit into the FP16 range and perform the backward pass as usual. After the all-reduce of the FP16 version of the gradients with respect to the weights we convert the gradients into FP32 and restore the original scale of the values before updating the weights.

In the beginning stages of training, the loss needs to be scaled down to avoid numerical overflow, while at the end of training, when the loss is small, we need to scale it up in order to avoid numerical underflow. Dynamic loss scaling takes care of both. It automatically scales down the loss when overflow is detected and since it is not possible to detect underflow, it scales the loss up if no overflows have been detected over the past 2,000 updates.

To evaluate training with lower precision, we first compare a baseline transformer model trained on 8 GPUs with 32-bit floating point (Our reimplementation) to the same model trained with 16-

[2] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

[3] SacreBLEU hash: `BLEU+case.mixed+lang.en-{de,fr}+numrefs.1+smooth.exp+test.wmt14/full+tok.13a+version.1.2.9`

model	# gpu	bsz	cumul	BLEU	updates	tkn/sec	time	speedup
Vaswani et al. (2017)	8×P100	25k	1	26.4	300k	~25k	~5,000	–
Our reimplementation	8×V100	25k	1	26.4	192k	54k	1,429	reference
+ `16-bit`	8	25k	1	26.7	193k	143k	495	2.9x
+ `cumul`	8	402k	16	26.7	13.7k	195k	447	3.2x
+ `2x lr`	8	402k	16	26.5	9.6k	196k	311	4.6x
+ `5k tkn/gpu`	8	365k	10	26.5	10.3k	202k	294	4.9x
16 nodes (from `+2x lr`)	128	402k	1	26.5	9.5k	1.53M	37	38.6x
+ `overlap comm+bwd`	128	402k	1	26.5	9.7k	1.82M	32	44.7x

Table 1: Training time (min) for reduced precision (`16-bit`), cumulating gradients over multiple backwards (`cumul`), increasing learning rate (`2x lr`) and computing each forward/backward with more data due to memory savings (`5k tkn/gpu`). Average time (excl. validation and saving models) over 3 random seeds to reach validation perplexity of 4.32 (2.11 NLL). Cumul=16 means a weight update after accumulating gradients for 16 backward computations, simulating training on 16 nodes. *WMT En-De, newstest13*.

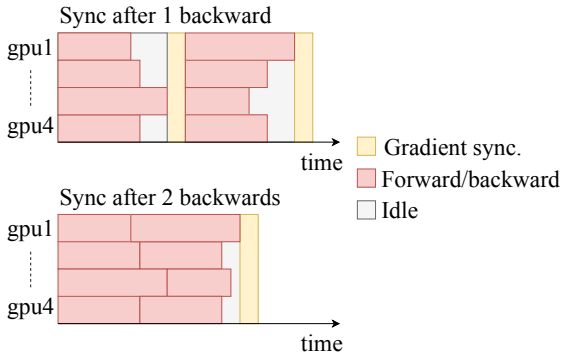

Figure 2: Accumulating gradients over multiple forward/backward steps speeds up training by: (i) reducing communication between workers, and (ii) saving idle time by reducing variance in workload between GPUs.

bit floating point (`16-bit`). Note, that we keep the batch size and other parameters equal. Table 1 reports training speed of various setups to reach validation perplexity 4.32 and shows that `16-bit` results in a 2.9x speedup.

4.2 Training with Larger Batches

Large batches are a prerequisite for distributed synchronous training, since it averages the gradients over all workers and thus the effective batch size is the sum of the sizes of all batches seen by the workers.

Figure 1 shows that bigger batches result in slower initial convergence when measured in terms of epochs (i.e. passes over the training set). However, when looking at the number of weight updates (i.e. optimization steps) large batches converge faster (Hoffer et al., 2017). These results support parallelization since the number of steps define the number of synchronization points for synchronous training.

Training with large batches is also possible on a single machine regardless of the number of GPUs or amount of available memory; one simply iterates over multiple batches and accumulates the resulting gradients before committing a weight update. This has the added benefit of reducing communication and reducing the variance in workload between different workers (see Figure 2), leading to a 36% increase in tokens/sec (Table 1, `cumul`). We discuss the issue of workload variance in more depth in Section 5.

Increased Learning Rate: Similar to Goyal et al. (2017) and Smith et al. (2018) we find that training with large batches enables us to increase the learning rate, which further shortens training time even on a single node (`2x lr`).

Memory Efficiency: Reduced precision also decreases memory consumption, allowing for larger sub-batches per GPU. We switch from a maximum of 3.5k tokens per GPU to a maximum of 5k tokens per GPU and obtain an additional 5% speedup (cf. Table 1; `2x lr` vs. `5k tkn/gpu`).

Table 1 reports our speed improvements due to reduced precision, larger batches, learning rate increase and increased per-worker batch size. Overall, we reduce training time from 1,429 min to 294 min to reach the same perplexity on the same hardware (8x NVIDIA V100), i.e. a 4.9x speedup.

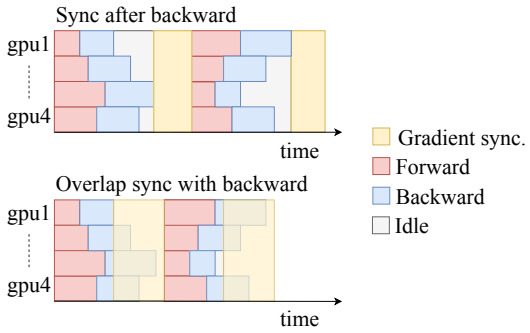

Figure 3: Illustration of how the backward pass in back-propagation can be overlapped with gradient synchronization to improve training speed.

4.3 Parallel Training

While large batch training improves training time even on a single node, another benefit of training with large batches is that it is easily parallelized across multiple nodes (machines). We run our previous 1-node experiment over 16 nodes of 8 GPUs each (NVIDIA V100), interconnected by Infiniband. Table 1 shows that with a simple, synchronous parallelization strategy over 16 nodes we can further reduce training time from 311 minutes to just 37 minutes (cf. Table 1; `2x lr` vs. `16 nodes`).

However, the time spent communicating gradients across workers increases dramatically when training with multiple nodes. In particular, our models contain over 200M parameters, therefore multi-node training requires transferring 400MB gradient buffers between machines. Fortunately, the sequential nature of back-propagation allows us to further improve multi-node training performance by beginning this communication in the background, while gradients are still being computed for the mini-batch (see Figure 3). Back-propagation proceeds sequentially from the top of the network down to the inputs. When the gradient computation for a layer finishes, we add the result to a synchronization buffer. As soon as the size of the buffer reaches a predefined threshold[4] we synchronize the buffered gradients in a background thread that runs concurrently with back-propagation down the rest of the network. Table 1 shows that by overlapping gradient communication with computation in the backwards pass, we can further reduce training time by 15%, from 37 minutes to just 32 minutes (cf. Table 1; `16 nodes` vs. `overlap comm+bwd`).

[4]We use a threshold of 150MB in this work.

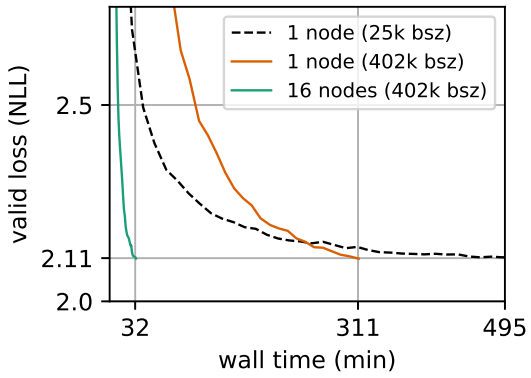

Figure 4: Validation loss (negative log likelihood on newstest13) versus training time on 1 vs 16 nodes.

	En–De	En–Fr
a. Gehring et al. (2017)	25.2	40.5
b. Vaswani et al. (2017)	28.4	41.0
c. Ahmed et al. (2017)	28.9	41.4
d. Shaw et al. (2018)	29.2	41.5
Our result	**29.3**	**43.2**
16-node training time	*85 min*	*512 min*

Table 2: BLEU on newstest2014 for WMT English-German (En–De) and English-French (En–Fr). All results are based on WMT'14 training data, except for En–De (b), (c), (d) and our result which are trained on WMT'16.

We illustrate the speedup achieved by large batches and parallel training in Figure 4.

4.4 Results with WMT Training Data

We report results on newstest14 for English-to-German (En-De) and English-to-French (En-Fr). For En-De, we train on the filtered version of WMT'16 from Vaswani et al. (2017). For En-Fr, we follow the setup of Gehring et al. (2017). In both cases, we train a "big" transformer on 16 nodes and average model parameters from the last 10 checkpoints (Vaswani et al., 2017). Table 2 reports 29.3 BLEU for En-De in 1h 25min and 43.2 BLEU for En-Fr in 8h 32min. We therefore establish a new state-of-the-art for both datasets, excluding settings with additional training data (Kutylowski, 2018). In contrast to Table 1, Table 2 reports times to convergence, not times to a specific validation likelihood.

Train set	En–De	En–Fr
WMT only	29.3	**43.2**
detok. SacreBLEU	*28.6*	*41.4*
16-node training time	*85 min*	*512 min*
WMT + Paracrawl	**29.8**	42.1
detok. SacreBLEU	*29.3*	*40.9*
16-node training time	*539 min*	*794 min*

Table 3: Test BLEU (*newstest14*) when training with WMT+Paracrawl data.

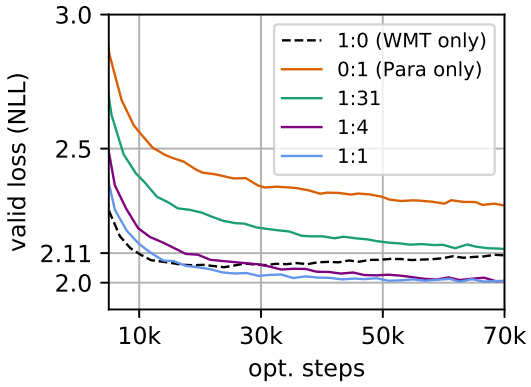

Figure 5: Validation loss when training on Paracrawl+WMT with varying sampling ratios. 1:4 means sampling 4 Paracrawl sentences for every WMT sentence. *WMT En-De, newstest13*.

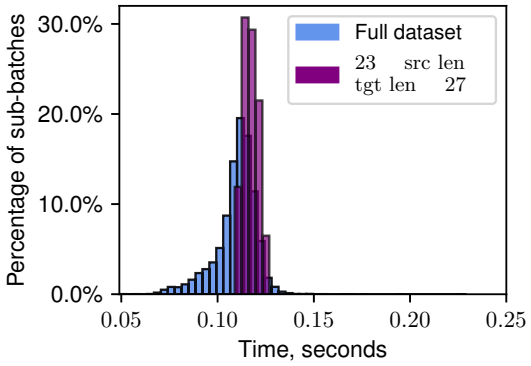

Figure 6: Histogram of time to complete one forward and backward pass for each sub-batch in the *WMT En-De* training dataset. Sub-batches consist of a variable number of sentences of similar length, such that each sub-batch contains at most 3.5k tokens.

4.5 Results with WMT & Paracrawl Training

Fast parallel training lets us additionally explore training over larger datasets. In this section we consider Paracrawl (ParaCrawl, 2018), a recent dataset of more than 4B parallel sentences for each language pair (En-De and En-Fr).

Previous work on Paracrawl considered training only on filtered subsets of less than 30M pairs (Xu and Koehn, 2017). We also filter Paracrawl by removing sentence-pairs with a source/target length ratio exceeding 1.5 and sentences with more than 250 words. We also remove pairs for which the source and target are copies (Ott et al., 2018). On En–De, this brings the set from 4.6B to 700M. We then train a En–De model on a clean dataset (WMT'14 news commentary) to score the remaining 700M sentence pairs, and retain the 140M pairs with best average token log-likelihood. To train an En–Fr model, we filter the data to 129M pairs using the same procedure.

Next, we explored different ways to weight the WMT and Paracrawl data. Figure 5 shows the validation loss for En-De models trained with different sampling ratios of WMT and filtered Paracrawl data during training. The model with 1:1 ratio performs best on the validation set, outperforming the model trained on only WMT data. For En-Fr, we found a sampling ratio of 3:1 (WMT:Paracrawl) performed best.

Test set results are given in Table 3. We find that Paracrawl improves BLEU on En–De to 29.8 but it is not beneficial for En–Fr, achieving just 42.1 vs. 43.2 BLEU for our baseline.

5 Analysis of Stragglers

In a distributed training setup with synchronized SGD, workers may take different amounts of time to compute gradients. Slower workers, or stragglers, cause other workers to wait. There are several reasons for stragglers but here we focus on the different amounts of time it takes to process the data on each GPU.

In particular, each GPU typically processes one *sub-batch* containing sentences of similar lengths, such that each sub-batch has at most N tokens (e.g., N = 3.5k tokens), with padding added as required. We refer to sub-batches as the data that is processed on each GPU worker whose combination is the entire batch. The sub-batches processed by a worker may therefore differ from other workers in the following three characteristics: the number of sentences, the maximum source sentence length, or the maximum target sentence length. To illustrate how these characteristics impact training

speed, Figure 6 shows the amount of time required to process the 44K sub-batches in the En-De training data. There is large variability in the amount time to process sub-batches with different characteristics: the mean time to process a sub-batch is 0.11 seconds, the slowest sub-batch takes 0.228 seconds and the fastest 0.049 seconds. Notably, there is much less variability if we only consider batches of a similar shape (e.g., batches where $23 \leq \text{src len} \approx \text{tgt len} \leq 27$).

Unsurprisingly, constructing sub-batches based on a maximum token budget as just described exacerbates the impact of stragglers. In Section 4.2 we observed that we could reduce the variance between workers by accumulating the gradients over multiple sub-batches on each worker before updating the weights (see illustration in Figure 2). A more direct, but naïve solution is to assign all workers sub-batches with a similar shape. However, this increases the variance of the gradients across batches and adversely affects the final model. Indeed, when we trained a model in this way, then it failed to converge to the target validation perplexity of 4.32 (cf. Table 1).

As an alternative, we construct sub-batches so that each one takes approximately the same amount of processing time across all workers. We first set a target for the amount of time a sub-batch should take to process (e.g., the 90th percentile in Figure 6) which we keep fixed across training. Next, we build a table to estimate the processing time for a sub-batch based on the number of sentences and maximum source and target sentence lengths. Finally, we construct each worker's sub-batches by tuning the number of sentences until the estimated processing time reaches our target. This approach improves single-node throughput from 143k tokens-per-second to 150k tokens-per-second, reducing the training time to reach 4.32 perplexity from 495 to 479 minutes (cf. Table 1, `16-bit`). Unfortunately, this is less effective than training with large batches, by accumulating gradients from multiple sub-batches on each worker (cf. Table 1, `cumul`, 447 minutes). Moreover, large batches additionally enable increasing the learning rate, which further improves training time (cf. Table 1, `2x lr`, 311 minutes).

6 Conclusions

We explored how to train state-of-the-art NMT models on large scale parallel hardware. We investigated lower precision computation, very large batch sizes (up to 400k tokens), and larger learning rates. Our careful implementation speeds up the training of a big transformer model (Vaswani et al., 2017) by nearly 5x on one machine with 8 GPUs.

We improve the state-of-the-art for WMT'14 En-Fr to 43.2 vs. 41.5 for Shaw et al. (2018), training in less than 9 hours on 128 GPUs. On WMT'14 En-De test set, we report 29.3 BLEU vs. 29.2 for Shaw et al. (2018) on the same setup, training our model in 85 minutes on 128 GPUs. BLEU is further improved to 29.8 by scaling the training set with Paracrawl data.

Overall, our work shows that future hardware will enable training times for large NMT systems that are comparable to phrase-based systems (Koehn et al., 2007). We note that multi-node parallelization still incurs a significant overhead: 16-node training is only ∼10x faster than 1-node training. Future work may consider better batching and communication strategies.

References

Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. 2017. Weighted transformer network for machine translation. *arxiv*, 1711.02132.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting distributed synchronous sgd. *Arxiv*, 1604.00981.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. *arxiv*, 1804.09849.

Adam Coates, Brody Huval, Tao Wang, David J. Wu, Bryan Catanzaro, and Andrew Y. Ng. 2013. Deep learning with cots hpc systems. In *Proc. of ICML*.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Training deep neural networks with low precision multiplications.

Jeff Dean. 2017. Machine learning for systems and systems for machine learning. In *Proc. of NIPS Workshop on ML Systems*.

Jeffrey Dean, Gregory S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proc. of NIPS*.

Ondrej Dusek and Filip Jurcícek. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *Proc. of ACL*.

Sergey Edunov, Myle Ott, and Sam Gross. 2017. Fairseq. https://github.com/pytorch/fairseq.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proc. of ICML*.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. In *Proc. of CVPR*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. In *Proc. of CVPR*.

Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Proc. of NIPS*, pages 1729–1739.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. of ICLR*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL Demo Session*.

Jaroslaw Kutylowski. 2018. Deepl press information. https://www.deepl.com/press.html.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *Proc. of ICLR*.

Myle Ott, Michael Auli, David Grangier, and MarcAurelio Ranzato. 2018. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning (ICML)*.

ParaCrawl. 2018. ParaCrawl. http://paracrawl.eu/download.html.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *Proc. of ICLR*.

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. 2017. Regularizing neural networks by penalizing confident output distributions. In *Proc. of ICLR Workshop*.

Matt Post. 2018. A call for clarity in reporting bleu scores. *arXiv*, 1804.08771.

Raul Puri, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro. 2018. Large scale language modeling: Converging on 40gb of text in four hours. *arXiv preprint arXiv:1808.01371*.

Jerry Quinn and Miguel Ballesteros. 2018. Pieces of eight: 8-bit neural machine translation. In *Proc. of NAACL*.

Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher Ré. 2018. High-accuracy low-precision training. *Arxiv*, 1803.03383.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proc. of ACL*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of ACL*.

Iulian Serban, Alessandro Sordoni, Ryan Joseph Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proc. of AAAI*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proc. of NAACL*.

Patrice Y. Simard and Hans Peter Graf. 1993. Backpropagation without multiplication. In *Proc. of NIPS*.

Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2018. Don't decay the learning rate, increase the batch size. In *Proc. of ICLR*.

Alessandro Sordoni, Michel Galley2, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Proc. of ACL*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *arXiv preprint arXiv:1512.00567*.

Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. 2017. Gradient Coding: Avoiding Stragglers in Distributed Learning. In *Proc. of ICML*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proc. of NIPS*.

Hainan Xu and Philipp Koehn. 2017. Zipporah: a fast and scalable data cleaning system for noisy web-crawled parallel corpora. In *Proc. of EMNLP*.

Min Ye and Emmanuel Abbe. 2018. Communication-Computation Efficient Gradient Coding. In *Proc. of ICML*.