PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 110 APRIL 2018 43–70

Training Tips for the Transformer Model

Martin Popel, Ondřej Bojar

Charles University, Faculty of Mathematics and Physics, Institute of Formal and Applied Linguistics,
Prague, Czechia

Abstract

This article describes our experiments in neural machine translation using the recent Tensor2Tensor framework and the Transformer sequence-to-sequence model (Vaswani et al., 2017). We examine some of the critical parameters that affect the final translation quality, memory usage, training stability and training time, concluding each experiment with a set of recommendations for fellow researchers. In addition to confirming the general mantra "more data and larger models", we address scaling to multiple GPUs and provide practical tips for improved training regarding batch size, learning rate, warmup steps, maximum sentence length and checkpoint averaging. We hope that our observations will allow others to get better results given their particular hardware and data constraints.

1. Introduction

It has been already clearly established that neural machine translation (NMT) is the new state of the art in machine translation, see e.g. the most recent evaluation campaigns (Bojar et al., 2017a; Cettolo et al., 2017). Many fundamental changes of the underlying neural network architecture are nevertheless still frequent and it is very difficult to predict which of the architectures has the best combination of properties to win in the long term, considering all relevant criteria like translation quality, model size, stability and speed of training, interpretability but also practical availability of good implementations. A considerable part of a model's success in translation quality consists in the training data, the model's sensitivity to noise in the data but also on a wide range of hyper-parameters that affect the training. Having the right setting of them turns out to be often a critical component for the success.

 Corresponding author: popel@ufal.mff.cuni.cz
Cite as: Martin Popel, Ondřej Bojar. Training Tips for the Transformer Model. The Prague Bulletin of Mathematical Linguistics No. 110, 2018, pp. 43–70. doi: 10.2478/pralin-2018-0002.

In this article, we experiment with a relatively new NMT model, called Transformer (Vaswani et al., 2017) as implemented in the Tensor2Tensor[1] (abbreviated T2T) toolkit, version 1.2.9. The model and the toolkit have been released shortly after the evaluation campaign at WMT2017[2] and its behavior on large-data news translation is not yet fully explored. We want to empirically explore some of the important hyper-parameters. Hopefully, our observations will be useful also for other researchers considering this model and framework.

While investigations into the effect of hyper-parameters like learning rate and batch size are available in the deep-learning community (e.g. Bottou et al., 2016; Smith and Le, 2017; Jastrzebski et al., 2017), these are either mostly theoretic or experimentally supported from domains like image recognition rather than machine translation. In this article, we fill the gap by focusing exclusively on MT and on the Transformer model only, providing hopefully the best practices for this particular setting.

Some of our observations confirm the general wisdom (e.g. larger training data are generally better) and quantify the behavior on English-to-Czech translation experiments. Some of our observations are somewhat surprising, e.g. that two GPUs are more than three times faster than a single GPU, or our findings about the interaction between maximum sentence length, learning rate and batch size.

The article is structured as follows. In Section 2, we discuss our evaluation methodology and main criteria: translation quality and speed of training. Section 3 describes our dataset and its preparations. Section 4 is the main contribution of the article: a set of commented experiments, each with a set of recommendations. Finally, Section 5 compares our best Transformer run with systems participating in WMT17. We conclude in Section 6.

2. Evaluation Methodology

Machine translation can be evaluated in many ways and some forms of human judgment should be always used for the ultimate resolution in any final application. The common practice in MT research is to evaluate the model performance on a test set against one or more human reference translations. The most widespread automatic metric is undoubtedly the BLEU score (Papineni et al., 2002), despite its acknowledged problems and better-performing alternatives (Bojar et al., 2017b). For simplicity, we stick to BLEU, too (we evaluated all our results also with chrF (Popović, 2015), but found no substantial differences from BLEU). In particular, we use the case-insensitive sacréBLEU[3] which uses a fixed tokenization (identical to `mteval-v14.pl --interna-`

[1] `https://github.com/tensorflow/tensor2tensor`

[2] `http://www.statmt.org/wmt17`

[3] `https://github.com/awslabs/sockeye/tree/master/contrib/sacrebleu`
The signature of the BLEU scores reported in this paper is `BLEU+case.lc+lang.en-cs+numrefs.1+smooth.exp+test.wmt13+tok.intl+version.1.2.3`.

`tional-tokenization`) and automatically downloads the reference translation for a given WMT testset.

2.1. Considerations on Stopping Criterion

The situation in NMT is further complicated by the fact that the training of NMT systems is usually non-deterministic,[4] and (esp. with the most recent models) hardly ever converges or starts overfitting[5] on reasonably big datasets. This leads to learning curves that never fully flatten let alone start decreasing (see Section 4.2). The common practice of machine learning to evaluate the model on a final test set when it started overfitting (or a bit sooner) is thus not applicable in practice.

Many papers in neural machine translation do not specify any stopping criteria whatsoever. Sometimes, they mention only an approximate number of days the model was trained for, e.g. Bahdanau et al. (2015), sometimes the exact number of training steps is given but no indication on "how much converged" the model was at that point, e.g. Vaswani et al. (2017). Most probably, the training was run until no further improvements were clearly apparent on the development test set, and the model was evaluated at that point. Such an approximate stopping criterion is rather risky: it is conceivable that different setups were stopped at different stages of training and their comparison is not fair.

A somewhat more reliable method is to keep training for a specified number of iterations or a certain number of epochs. This is however not a perfect solution either, if the models are not quite converged at that time and the difference in their performance is not sufficiently large. It is quite possible that e.g. a more complex model would need a few more epochs and eventually arrived at a higher score than its competitor. Also, the duration of one training step (or one epoch) differs between models (see Section 4.1) and from the practical point of view, we are mostly interested in the wall-clock time.

When we tried the standard technique of early stopping, when N subsequent evaluations on the development test set do not give improvements larger than a given delta, we saw a big variance in the training time and final BLEU, even for experiments with the same hyper-parameters and just a different random seed. Moreover to get the best results, we would have had to use a very large N and a very small delta.

[4] Even if we fix the random seed (which was not done properly in T2T v1.2.9), a change of some hyper-parameters may affect the results not because of the change itself, but because it influenced the random initialization.

[5] By overfitting we mean here that the translation quality (test-set BLEU) begins to worsen, while the training loss keeps improving.

2.2. Our Final Choice: Full Learning Curves

Based on the discussion above, we decided to report always the full learning curves and not just single scores. This solution does not fully prevent the risk of premature judgments, but the readers can at least judge for themselves if they would expect any sudden twist in the results or not.

In all cases, we plot the case-insensitive BLEU score against the wall-clock time in hours. This solution obviously depends on the hardware chosen, so we always used the same equipment: one up to eight GeForce GTX 1080 Ti GPUs with NVIDIA driver 375.66. Some variation in the measurements is unfortunately unavoidable because we could not fully isolate the computation from different processes on the same machine and from general network traffic, but based on our experiments with replicated experiments such variation is negligible.

2.3. Terminology

For clarity, we define the following terms and adhere to them for the rest of the paper:

Translation quality is an automatic estimate of how well the translation carried out by a particular fixed model expresses the meaning of the source. We estimate translation quality solely by BLEU score against one reference translation.

Training Steps denote the number of iterations, i.e. the number of times the optimizer update was run. This number also equals the number of (mini)batches that were processed.

Batch Size is the number of training examples used by one GPU in one training step. In sequence-to-sequence models, batch size is usually specified as the number of *sentence pairs*. However, the parameter `batch_size` in T2T translation specifies the approximate number of *tokens* (subwords) in one batch.[6] This allows to use a higher number of short sentences in one batch or a smaller number of long sentences.

Effective Batch Size is the number of training examples consumed in one training step. When training on multiple GPUs, the parameter `batch_size` is interpreted per GPU. That is, with `batch_size=1500` and 8 GPUs, the system actually digests 12k subwords of each language in one step.

Training Epoch corresponds to one complete pass over the training data. Unfortunately, it is not easy to measure the number of training epochs in T2T.[7] T2T

[6] For this purpose, the number of tokens in a sentence is defined as the maximum of source and target subwords. T2T also does reordering and bucketing of the sentences by their length to minimize the use of padding symbols. However, some padding is still needed, thus `batch_size` only approximates the actual number of (non-padding) subwords in a batch.

[7] `https://github.com/tensorflow/tensor2tensor/issues/415`

reports only the number of training steps. In order to convert training steps to epochs, we need to multiply the steps by the effective batch size and divide by the number of subwords in the training data (see Section 3.1). The segmentation of the training data into subwords is usually hidden to the user and the number of subwords must be thus computed by a special script.

Computation Speed is simply the observed number of training steps per hour. Computation speed obviously depends on the hardware (GPU speed, GPU-CPU communication) and software (driver version, CUDA library version, implementation). The main parameters affecting computation speed are the model size, optimizer and other settings that directly modify the formula of the neural network.

Training Throughput is the amount of training data digested by the training. We report training throughput in subwords per hour. Training Throughput equals to the Computation Speed multiplied by the effective batch size.

Convergence Speed or **BLEU Convergence** is the increase in BLEU divided by time. Convergence speed changes heavily during training, starting very high and decreasing as the training progresses. A converged model should have convergence speed of zero.

Time Till Score is the training time needed to achieve a certain level of translation quality, in our case BLEU. We use this as an informal measure because it is not clear how to define the moment of "achieving" a given BLEU score. We define it as time after which the BLEU never falls below the given level.[8]

Examples Till Score is the number of training examples (in subwords) needed to achieve a certain level of BLEU. It equals to the Time Till Score multiplied by Training Throughput.

2.4. Tools for Evaluation within Tensor2Tensor

T2T, being implemented in TensorFlow, provides nice TensorBoard visualizations of the training progress. The original implementation was optimized towards speed of evaluation rather than towards following the standards of the field. T2T thus reports "approx-bleu" by default, which is computed on the internal subwords (never exposed to the user, actually) instead of words (according to BLEU tokenization). As a result, "approx-bleu" is usually about 1.2–1.8 times higher than the real BLEU. Due to its dependence on the training data (for the subword vocabulary), it is not easily reproducible in varying experiments and thus not suitable for reporting in publications.

[8] Such definition of Time Till Score leads to a high variance of its values because of the relatively high BLEU variance between subsequent checkpoints (visible as a "flickering" of the learning curves in the figures). To decrease the variation one can use a bigger development test set.

	sentences	EN words	CS words
CzEng 1.7	57 M	618 M	543 M
europarl-v7	647 k	15 M	13 M
news-commentary-v11	190 k	4.1 M	3.7 M
commoncrawl	161 k	3.3 M	2.9 M
Total	58 M	640 M	563 M

Table 1: Training data resources

We implemented a helper script `t2t-bleu` which computes the "real" BLEU (giving the same result as sacréBLEU with `--tokenization intl`). Our script can be used in two ways:

- To evaluate one translated file:
  `t2t-bleu --translation=my-wmt13.de --reference=wmt13_deen.de`
- To evaluate all translations in a given directory (created e.g. by `t2t-translate-all`) and store the results in a TensorBoard events file. All the figures in this article were created this way.

We also implemented `t2t-translate-all` and `t2t-avg-all` scripts, which translate all checkpoints in a given directory and average a window of N subsequent checkpoints, respectively.[9] For details on averaging see Section 4.10.

3. Data Selection and Preprocessing

We focused on the English-to-Czech translation direction. Most of our training data comes from the CzEng parallel treebank, version 1.7 (57M sentence pairs),[10] and the rest (1M sentence pairs) comes from three smaller sources (Europarl, News Commentary, Common Crawl) as detailed in Table 1.

We use this dataset of 58M sentence pairs for most our experiments. In some experiments (in Sections 4.2 and 4.6), we substitute CzEng 1.7 with an older and considerably smaller CzEng 1.0 (Bojar et al., 2012) containing 15M sentence pairs (233M/206M of en/cs words).

To plot the performance throughout the training, we use WMT newstest2013 as a development set (not overlapping with the training data). In Section 5, we apply our best model (judged from the performance on the development set) to the WMT newstest2017, for comparison with the state-of-the-art systems.

[9] All three scripts are now merged in the T2T master. All three scripts can be used while the training is still in progress, i.e. they wait a given number of minutes for new checkpoints to appear.

[10] `http://ufal.mff.cuni.cz/czeng/czeng17`, which is a subset of CzEng 1.6 (Bojar et al., 2016).

3.1. Training Data Preprocessing

Data preprocessing such as tokenization and truecasing has always been a very important part of the setup of statistical machine translation systems. A huge leap in scaling NMT to realistic data size has been achieved by the introduction of subword units (Sennrich et al., 2016), but the long-term vision of the deep-learning community is to leave all these "technicalities" up to the trained neural network and feed it with as original input as possible (see e.g. Lee et al., 2016).

T2T adopts this vision and while it supports the use of external subword units, it comes with its own built-in method similar to the word-piece algorithm by Wu et al. (2016) and does not expect the input to be even tokenized. Based on a small sample of the training data, T2T will train a subword vocabulary and apply it to all the training and later evaluation data.

We follow the T2T default and provide raw plain text training sentences. We use the default parameters: shared source and target (English and Czech) subword vocabulary of size 32k.[11] After this preprocessing, the total number of subwords in our main training data is 992 millions (taking the maximum of English and Czech lengths for each sentence pair, as needed for computing the number of epochs, see Section 2.3). The smaller dataset CzEng 1.0 has 327 million subwords. In both cases the average number of subwords per (space-delimited) word is about 1.5.

Even when following the defaults, there are some important details that should be considered. We thus provide our first set of technical tips here:

Tips on Training Data Preprocessing

- Make sure that the subword vocabulary is trained on a sufficiently large sample of the training data.[12]
- As discussed in Section 4.5, a higher batch size may be beneficial for the training and the batch size can be higher when excluding training sentences longer than a given threshold. This can be controlled with parameter `max_length` (see Section 4.4), but it may be a good idea to exclude too long sentences even before preparing the training data using `t2t-datagen`. This way the TFRecords training files will be smaller and their processing a bit faster.[13]

[11] More details on T2T with BPE subword units by Sennrich et al. (2016) vs. the internal implementation can be found in the technical report "Morphological and Language-Agnostic Word Segmentation for NMT" attached to the Deliverable 2.3 of the project QT21: `http://www.qt21.eu/resources/`.

[12] This is controlled by a `file_byte_budget` constant, which must be changed directly in the source code in T2T v1.2.9. A sign of too small training data for the subword vocabulary is that the `min_count` as reported in the logs is too low, so the vocabulary is estimated from words seen only once or twice.

[13] We did no such pre-filtering in our experiments.

4. Experiments

In this section, we present several experiments, always summarizing the observations and giving some generally applicable tips that we learned. All experiments were done with T2T v1.2.9 unless stated otherwise.

We experiment with two sets of hyper-parameters pre-defined in T2T: `transformer_big_single_gpu` (BIG) and `transformer_base_single_gpu` (BASE), which differ mainly in the size of the model. Note that `transformer_big_single_gpu` and `transformer_base_single_gpu` are just names of a set of hyper-parameters, which can be applied even when training on multiple GPUs, as we do in our experiments, see Section 4.7.[14]

Our baseline setting uses the BIG model with its default hyper-parameters except for:

- `batch_size=1500` (see the discussion of different sizes in Section 4.5),
- `--train_steps=6000000`, i.e. high enough, so we can stop each experiment manually as needed,
- `--save_checkpoints_secs=3600` which forces checkpoint saving each hour (see Section 4.10),
- `--schedule=train` which disables the internal evaluation with `approx_bleu` and thus makes training a bit faster (see Section 2).[15]

4.1. Computation Speed and Training Throughput

We are primarily interested in the translation quality (BLEU learning curves and Time Till Score) and we discuss it in the following sections 4.2–4.10. In this section, we focus however only on the *computation speed* and *training throughput*. Both are affected by three important factors: batch size, number of used GPUs and model size. The speed is usually almost constant for a given experiment.[16]

Table 2 shows the computation speed and training throughput for a single GPU and various batch sizes and model sizes (BASE and BIG). The BASE model allows for using a higher batch size than the BIG model. The cells where the BIG model resulted in out-of-memory errors are marked with "OOM".[17] We can see that the computa-

[14] According to our experiments (not reported here), `transformer_big_single_gpu` is better than `transformer_big` even when training on 8 GPUs, although the naming suggests that the T2T authors had an opposite experience.

[15] Also there are some problems with the alternative schedules `train_and_evaluate` (it needs more memory) and `continuous_train_and_eval` (see `https://github.com/tensorflow/tensor2tensor/issues/556`).

[16] TensorBoard shows `global_step/sec` statistics, i.e. the computation speed curve. These curves in our experiments are almost constant for the whole training with variation within 2%, except for moments when a checkpoint is being saved (and the computation speed is thus much slower).

[17] For these experiments, we used `max_length=50` in order to be able to test bigger batch sizes. However, in additional experiments we checked that `max_length` does not affect the training throughput itself.

	model	
batch_size	BASE	BIG
500	43.4k	23.6k
1000	30.2k	13.5k
1500	22.3k	9.8k
2000	16.8k	7.5k
2500	14.4k	6.5k
3000	12.3k	OOM
4500	8.2k	OOM
6000	6.6k	OOM

(a) Computation speed (steps/hour)

	model	
batch_size	BASE	BIG
500	21.7M	11.9M
1000	30.2M	13.5M
1500	33.4M	14.7M
2000	33.7M	15.0M
2500	36.0M	16.2M
3000	37.0M	OOM
4500	36.7M	OOM
6000	39.4M	OOM

(b) Training throughput (subwords/hour)

Table 2: Computation speed and training throughput for a single GPU.

tion speed decreases with increasing batch size because not all operations in GPU are fully batch-parallelizable. The training throughput grows sub-linearly with increasing batch size, so based on these experiments only, there is just a small advantage when setting the batch size to the maximum value. We will return to this question in Section 4.5, while taking into account the translation quality.

We can also see the BASE model has approximately two times bigger throughput as well as computation speed relative to the BIG model.

GPUs	steps/hour	subwords/hour
1	9.8k	14.7M
2	7.4k	22.2M
6	5.4k	48.6M
8	5.6k	67.2M

Table 3: Computation speed and training throughput for various numbers of GPUs, with the BIG model and `batch_size=1500`.

Table 3 uses the BIG model and `batch_size=1500`, while varying the number of GPUs. The overhead in GPU synchronization is apparent from the decreasing computation speed. Nevertheless, the training throughput still grows with more GPUs, so e.g. with 6 GPUs we process 3.2 times more training data per hour relative to a single GPU (while without any overhead we would hypothetically expect 6 times more data).

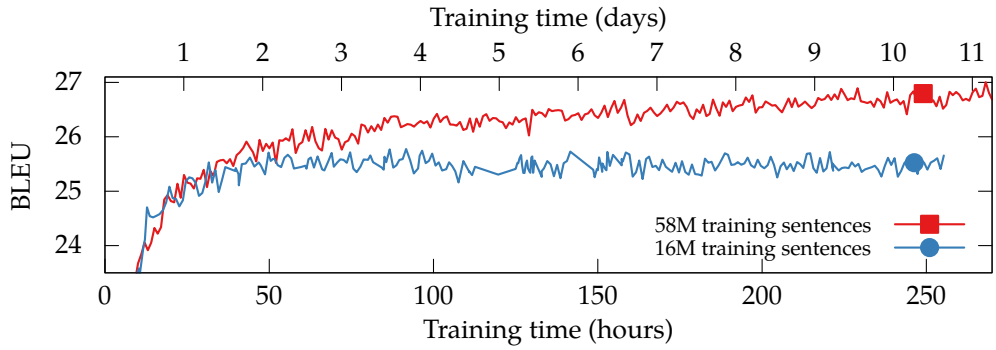

Figure 1: Training data size effect. BLEU learning curves for our main training dataset with 58 million sentence pairs and an alternative training dataset with 16 million sentence pairs. Both trained with 8 GPUs, BIG model and `batch_size=1500`.

The overhead when scaling to multiple GPUs is smaller than the overhead when scaling to a higher batch size. Scaling from a single GPU to 6 GPUs increases the throughput 3.2 times, but scaling from batch size 1000 to 6000 on a single GPU increases the throughput 1.3 times.

4.2. Training Data Size

For this experiment, we substituted CzEng 1.7 with CzEng 1.0 in the training data, so the total training size is 16 million sentence pairs (255M / 226M of English/Czech words). Figure 1 compares the BLEU learning curves of two experiments which differ only in the training data: the baseline CzEng 1.7 versus the smaller CzEng 1.0. Both are trained on the same hardware with the same hyper-parameters (8 GPUs, BIG, `batch_size=1500`). Training on the smaller dataset (2.5 times smaller in the number of words) converges to BLEU of about 25.5 after two days of training and does not improve over the next week of training. Training on the bigger dataset gives slightly worse results in the first eight hours of training (not shown in the graph) but clearly better results after two days of training, reaching over 26.5 BLEU after eight days.[18]

With `batch_size=1500` and 8 GPUs, training one epoch of the smaller dataset (with CzEng 1.0) takes 27k steps (5 hours of training), compared to 83k steps (15 hours) for the bigger dataset (with CzEng 1.7). This means *about 10 epochs in the smaller dataset were needed for reaching the convergence* and this is also the moment when the bigger

[18] We compared the two datasets also in another experiment with two GPUs, where CzEng 1.7 gave slightly worse results than CzEng 1.0 during the first two days of training but clearly better results after eight days. We hypothesize CzEng 1.0 is somewhat cleaner than CzEng 1.7.

dataset starts being clearly better. However, *even 18 epochs in the bigger dataset were not enough to reach the convergence*. *enough to reach the convergence*

Tips on Training Data Size

- For comparing different datasets (e.g. smaller and cleaner vs. bigger and noisier), we need to train long enough because *results after first hours (or days if training on a single GPU) may be misleading*.
- For large training data (as CzEng 1.7 which has over half a gigaword), *BLEU improves even after one week of training on eight GPUs* (or after 20 days of training on two GPUs in another experiment).
- *We cannot easily interpolate one dataset results to another dataset.* While the smaller training data (with CzEng 1.0) converged after 2 days, the main training data (with CzEng 1.7), which is 2.5 times bigger, continues improving even after 2.5×2 days.[19]

4.3. Model Size

Choosing the right model size is important for practical reasons: larger models may not fit any more on your GPU or they may require to use a very small batch size.

We experiment with two models,[20] as pre-defined in Tensor2Tensor – `transformer_big_single_gpu` (BIG) and `transformer_base_single_gpu` (BASE), which differ in four hyper-parameters summarized in Table 4.

model	hidden_size	filter_size	num_heads	adam_beta2
BASE	512	2048	8	0.980
BIG	1024	4096	16	0.998

Table 4: `transformer_big_single_gpu` (BIG) and `transformer_base_single_gpu` (BASE) hyper-parameter differences.

Figure 2 shows that on a single GPU, the BIG model becomes clearly better than the BASE model after 4 hours of training if we keep the batch size the same – 2000 (and we have confirmed it with 1500 in other experiments). However, the BASE model takes less memory, so we can afford a higher batch size, in our case 4500 (with no `max_length` restriction, see the next section), which improves the BLEU (see Section 4.5). But even

[19] Although such an expectation may seem naïve, we can find it in literature. For example, Bottou (2012) in Section 4.2 writes: "*Expect the validation performance to plateau after a number of epochs roughly comparable to the number of epochs needed to reach this point on the small training set.*"

[20] We tried also a model three times as large as BASE (1.5 times as large as BIG), but it did not reach better results than BIG, so we don't report it here.

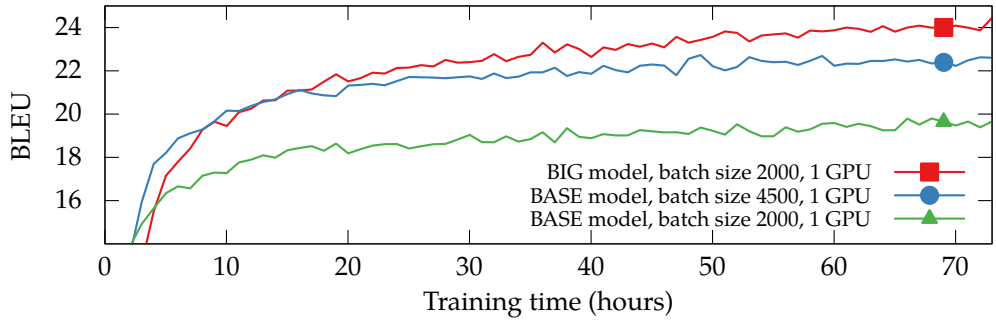

Figure 2: Effect of model size and batch size on a single GPU.

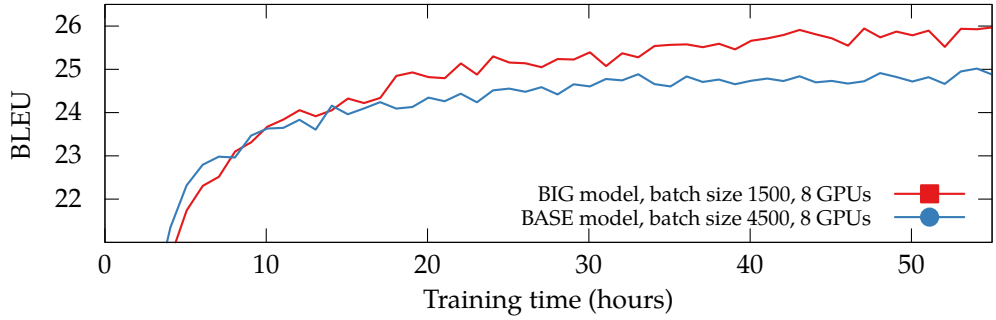

Figure 3: Effect of model size and batch size on 8 GPUs.

so, after less than one day of training, BIG with batch size 2000 becomes better than BASE with batch size 4500 (or even 6000 with `max_length=70` in another experiment) and the difference grows up to 1.8 BLEU after three days of training.

Figure 3 confirms this with 8 GPUs – here BIG with batch size 1500 becomes clearly better than BASE with batch size 4500 after 18 hours of training.

Tips on Model Size

- *Prefer the BIG over the BASE model* if you plan to train longer than one day and have 11 GB (or more) memory available on GPU.
- With less memory you should benchmark BIG and BASE with the maximum possible batch size.

	maximum batch size			longer sentences	
max_length	BIG+Adam	BIG+Adafactor	BASE+Adam	train	test
none	2040	2550	4950	0.0%	0.0%
150	2230	2970	5430	0.2%	0.0%
100	2390	3280	5990	0.7%	0.3%
70	2630	3590	6290	2.1%	2.2%
50	2750	3770	6430	5.0%	9.1%

Table 5: Maximum batch size which fits into 11GB memory for various combinations of `max_length` (maximum sentence length in subwords), model size (base or big) and optimizer (Adam or Adafactor). The last two columns show the percentage of sentences in the train (CzEng 1.7) and test (wmt13) data that are longer than a given threshold.

- For fast debugging (of model-size-unrelated aspects) use a model called `transformer_tiny`.

4.4. Maximum Training Sentence Length

The parameter `max_length` specifies the maximum length of a sentence in subwords. Longer sentences (either in source or target language) are excluded from the training completely. If no `max_length` is specified (which is the default), `batch_size` is used instead. Lowering the `max_length` allows to use a higher batch size or a bigger model. Since the Transformer implementation in T2T can suddenly run out of memory even after several hours of training, it is good to know how large batch size fits in your GPU. Table 5 presents what we empirically measured for the BASE and BIG models with Adam and Adafactor[21] optimizers and various `max_length` values.

Setting `max_length` too low would result in excluding too many training sentences and biasing the translation towards shorter sentences, which would hurt the translation quality. The last two columns in Table 5 show that setting `max_length` to 70 (resp. 100) results in excluding only 2.1% (resp. 0.7%) of sentences in the training data, and only 2.2% (resp. 0.3%) sentences in the development test data are longer, so the detrimental effect of smaller training data and length bias should be minimal in this setting. However, our experiments with `batch_size`=1500 in Figure 4 show a strange drop in BLEU after one hour of training for all experiments with `max_length` 70 or lower. Even with `max_length` 150 or 200 the BLEU learning curve is worse than with `max_length`=400, which finally gives the same result as not using any `max_length`

[21] The Adafactor optimizer (Shazeer and Stern, 2018) is available only in T2T 1.4.2 or newer and has three times smaller models than Adam because it does not store first and second moments for all weights. We leave further experiments with Adafactor for future work.

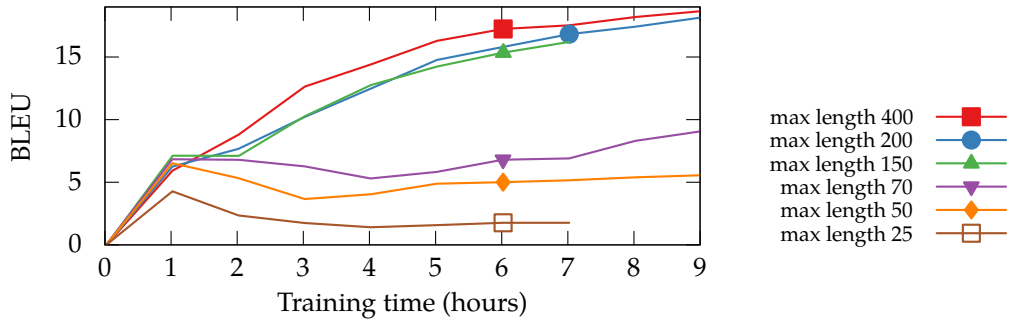

Figure 4: Effect of restricting the training data to various `max_length` values. All trained on a single GPU with the BIG model and `batch_size`=1500. An experiment without any `max_length` is not shown, but it has the same curve as `max_length`=400.

restriction. The training loss of `max_length`=25 (and 50 and 70) has high variance and stops improving after the first hour of training but shows no sudden increase (as in the case of diverged training discussed in Section 4.6 when the learning rate is too high). We have no explanation for this phenomenon.[22]

We did another set of experiments with varying `max_length`, but this time with `batch_size`=2000 instead of 1500. In this case, `max_length` 25 and 50 still results in slower growing BLEU curves, but 70 and higher has the same curve as no `max_length` restriction. So in our case, *if the batch size is high enough, the `max_length` has almost no effect on BLEU*, but this should be checked for each new dataset.

We trained several models with various `max_length` for three days and observed that *they are not able to produce longer translations than what was the maximum length used in training*, even if we change the decoding parameter alpha.

Tips on `max_length`

- *Set (a reasonably low) `max_length`.* This allows to use a higher batch size and prevents out-of-memory errors after several hours of training. Also, with a higher percentage of training sentences that are almost `max_length` long, there is a higher chance that the training will fail either immediately (if the batch size is too high) or never (otherwise).,
- *Set a reasonably high `max_length`.* Consider the percentage of sentences excluded from training and from the targeted development test set and also watch for unexpected drops (or stagnations) of the BLEU curve in the first hours of training.

[22] `https://github.com/tensorflow/tensor2tensor/issues/582`

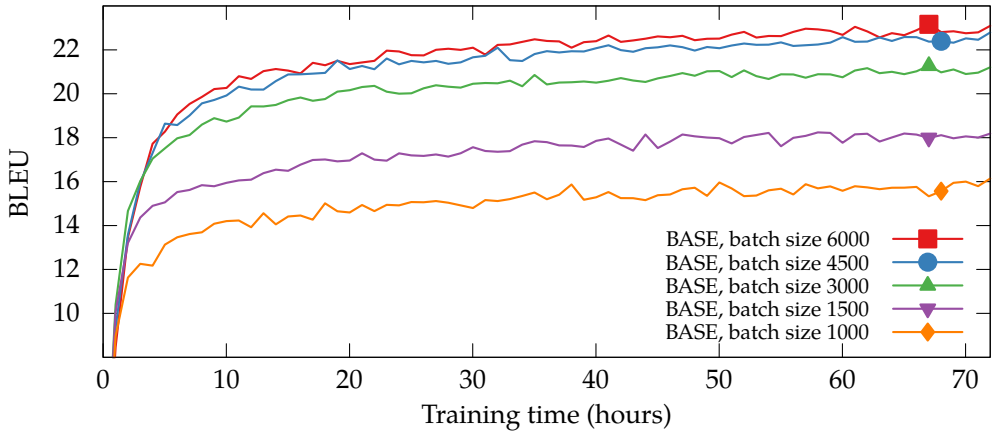

Figure 5: Effect of the batch size with the BASE model. All trained on a single GPU.

4.5. Batch Size

The default `batch_size` value in recent T2T versions is 4096 subwords for all models except for `transformer_base_single_gpu`, where the default is 2048. However, we recommend to always set the batch size explicitly[23] or at least make a note what was the default in a given T2T version when reporting experimental results.

Figure 5 shows learning curves for five different batch sizes (1000, 1500, 3000, 4500 and 6000) for experiments with a single GPU and the BASE model.[24] A higher batch size *up to 4500* is clearly better in terms of BLEU as measured by Time Till Score and Examples Till Score metrics defined in Section 4.1. For example, to get over BLEU of 18 with `batch_size`=3000, we need 7 hours (260M examples), and with `batch_size`=1500, we need about 3 days (2260M examples) i.e. 10 times longer (9 time more examples). From Table 2a we know that bigger batches have slower computation speed, so when re-plotting Figure 5 with steps instead of time on the x-axis, the difference between the curves would be even bigger. From Table 2b we know that bigger batches have slightly higher training throughput, so when re-plotting with number of examples processed on the x-axis, the difference will be smaller, but still visible. The only exception is the difference between batch size 4500 and 6000, which is very small and can be fully

[23] e.g. `--hparams="batch_size=1500,learning_rate=0.20,learning_rate_warmup_steps=16000"`
As the batch size is specified in subwords, we see no advantage in using power-of-two values.

[24] All the experiments in Figure 5 use `max_length`=70, but we have got the same curves when re-running without any `max_length` restrictions, except for `batch_size`=6000 which failed with OOM.

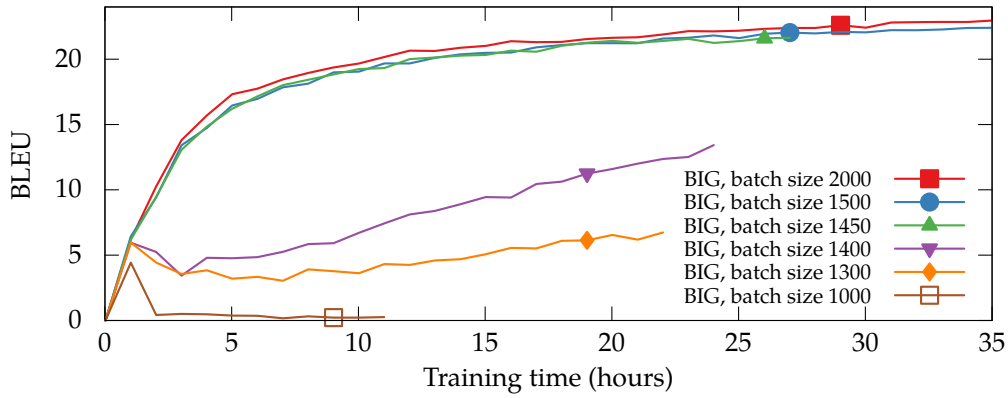

Figure 6: Effect of the batch size with the BIG model. All trained on a single GPU.

explained by the fact that batch size 6000 has 7% higher throughput than batch size 4500.

So *for the BASE model, a higher batch size gives better results*, although with diminishing returns. This observation goes against the common knowledge in other NMT frameworks and deep learning in general (Keskar et al., 2017) that smaller batches proceed slower (training examples per hour) but result in better generalization (higher test-set BLEU) in the end. In our experiments with the BASE model in T2T, bigger batches are not only faster in training throughput (as could be expected), but also faster in convergence speed, Time Till Score and Examples Till Score.

Interestingly, when replicating these experiments *with the BIG model, we see quite different results*, as shown in Figure 6. The BIG model needs a certain minimal batch size to start converging at all, but for higher batch sizes there is almost no difference in the BLEU curves (but still, bigger batch never makes the BLEU worse in our experiments). In our case, the sharp difference is between batch size 1450, which trains well, and 1400, which drops off after two hours of training, recovering only slowly.

According to Smith and Le (2017) and Smith et al. (2017), the *gradient noise scale*, i.e. scale of random fluctuations in the SGD (or Adam etc.) dynamics, is proportional to learning rate divided by the batch size (cf. Section 4.8). Thus when lowering the batch size, we increase the noise scale and the training may *diverge*. This may be either permanent, as in the case of batch size 1000 in Figure 6, or temporary, as in the case of batch size 1300 and 1400, where the BLEU continues to grow after the temporary drop, but much more slowly than the non-diverged curves.

We are not sure what causes the difference between the BASE and BIG models with regards to the sensitivity to batch size. One hypothesis is that the BIG model is more

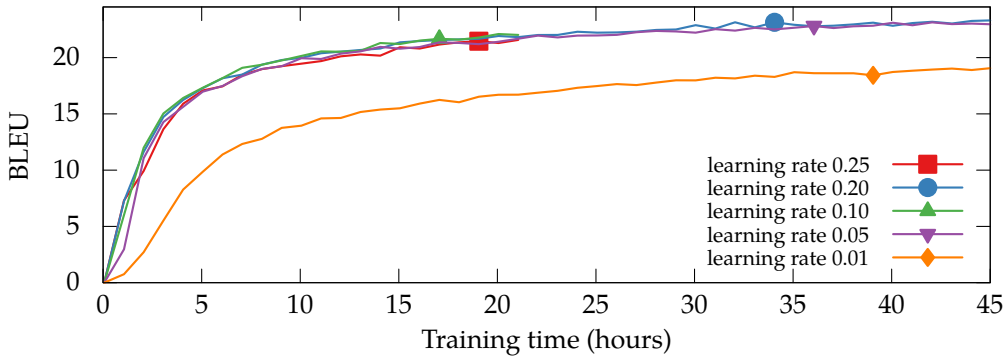

Figure 7: Effect of the learning rate on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and warmup steps (16k).

difficult to initialize and thus more sensitive to divergence in the early training phase. Also while for BASE, increasing the batch size was highly helpful until 4500, for BIG this limit may be below 1450, i.e. below the minimal batch size needed for preventing diverged training.

Tip on Batch Size

- *Batch size should be set as high as possible* while keeping a reserve for not hitting the out-of-memory errors. It is advisable to establish the largest possible batch size before starting the main and long training.

4.6. Learning Rate and Warmup Steps on a Single GPU

The default learning rate in T2T translation models is 0.20. Figure 7 shows that varying the value within range 0.05–0.25 makes almost no difference. Setting the learning rate too low (0.01) results in notably slower convergence. Setting the learning rate too high (0.30, not shown in the figure) results in *diverged* training, which means in this case that the learning curve starts growing as usual, but at one moment drops down almost to zero and stays there forever.

A common solution to prevent diverged training is to decrease the `learning_rate` parameter or increase `learning_rate_warmup_steps` or introduce gradient clipping. The `learning_rate_warmup_steps` parameter configures a `linear_warmup_rsqrt_decay` schedule[25] and it is set to 16 000 by default (for the BIG model), meaning that within

[25] The schedule was called `noam` in T2T versions older than 1.4.4.

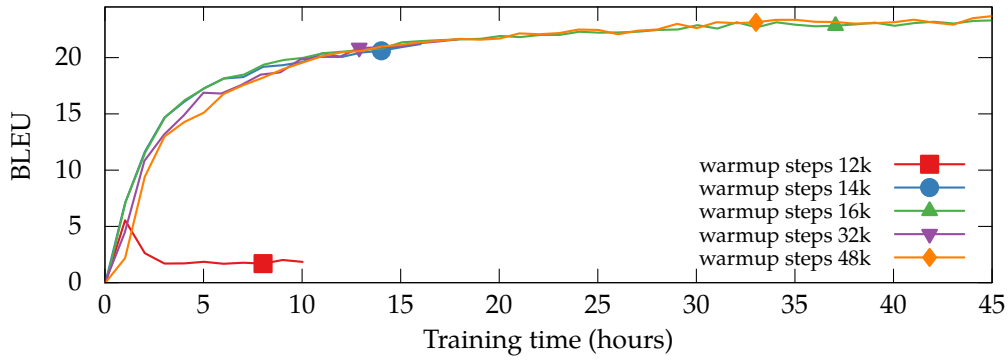

Figure 8: Effect of the warmup steps on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and learning rate (0.20).

the first 16k steps the learning rate grows linearly and then follows an inverse square root decay ($t^{-0.5}$, cf. Section 4.8.3). At 16k steps, the actual learning rate is thus the highest.

If a divergence is to happen, it usually happens within the first few hours of training, when the actual learning rate becomes the highest. Once we increased the warmup steps from 16k to 32k, we were able to train with the learning rate of 0.30 and even 0.50 without any divergence. The learning curves looked similarly to the baseline one (with default values of 16k warmup steps and learning rate 0.20). When trying learning rate 1.0, we had to increase warmup steps to 60k (with 40k the training diverged after one hour) – this resulted in a slower convergence at first (about 3 BLEU lower than the baseline after 8 hours of training), but after 3–4 days of training having the same curve as the baseline.

Figure 8 shows the effect of different warmup steps with a fixed learning rate (the default 0.20). Setting warmup steps too low (12k) results in diverged training. Setting them too high (48k, green curve) results in a slightly slower convergence at first, but matching the baseline after a few hours of training.

We can conclude that for a single GPU and the BIG model, there is a relatively large range of learning rate and warmup steps values that achieve the optimal results. The default values `learning_rate`=0.20 and `learning_rate_warmup_steps`=16000 are within this range.

Tips on Learning Rate and Warmup Steps

- *In case of diverged training, try gradient clipping and/or more warmup steps.*

- If that does not help (or if the warmup steps are too high relative to the expected total training steps), try decreasing the learning rate.
- Note that when you decrease warmup steps (and keep learning rate), you also increase the maximum actual learning rate because of the way how the `linear_warmup_rsqrt_decay` (aka `noam`) schedule is implemented.[26]

4.7. Number of GPUs

T2T allows to train with multiple GPUs on the same machine simply using the parameter `--worker_gpus`.[27] As explained in Section 2.3, the parameter `batch_size` is interpreted per GPU, so with 8 GPUs, the *effective batch size* is 8 times bigger.

A single-GPU experiment with batch size 4000, should give exactly the same results as two GPUs and batch size 2000 and as four GPUs and batch size 1000 because the effective batch size is 4000 in all three cases. We have confirmed this empirically. By the "same results" we mean BLEU (or train loss) versus training steps on the x-axis. When considering time, the four-GPU experiment will be the fastest one, as explained in Section 4.1.

Figure 9 shows BLEU curves for different numbers of GPUs and the BIG model with batch size, learning rate and warmup steps fixed on their default values (1500, 0.20 and 16k, respectively). As could be expected, training with more GPUs converges faster. What is interesting is the Time Till Score. Table 6 lists the approximate training time and number of training examples (in millions of subwords) needed to "surpass" (i.e. achieve and never again fall below) BLEU of 25.6.

# GPUs	hours	subwords (M)
1	> 600	> 9000
2	203	$2322 \cdot 2 = 4644$
6	56	$451 \cdot 6 = 2706$
8	40	$341 \cdot 8 = 2728$

Table 6: Time and training data consumed to reach BLEU of 25.6, i.e. Time Till Score and Examples Till Score. Note that the experiment on 1 GPU was ended after 25 days of training without clearly surpassing the threshold (already outside of Figure 9).

[26] This holds at least in T2T versions 1.2.9–1.5.2, but as it is somewhat unexpected/unintuitive for some users, it may be fixed in future, see `https://github.com/tensorflow/tensor2tensor/issues/517`.

[27] and making sure environment variable `CUDA_VISIBLE_DEVICES` is set so enough cards are visible. T2T allows also distributed training (on multiple machines), but we have not experimented with it. Both single-machine multi-gpu and distributed training use synchronous Adam updates by default.

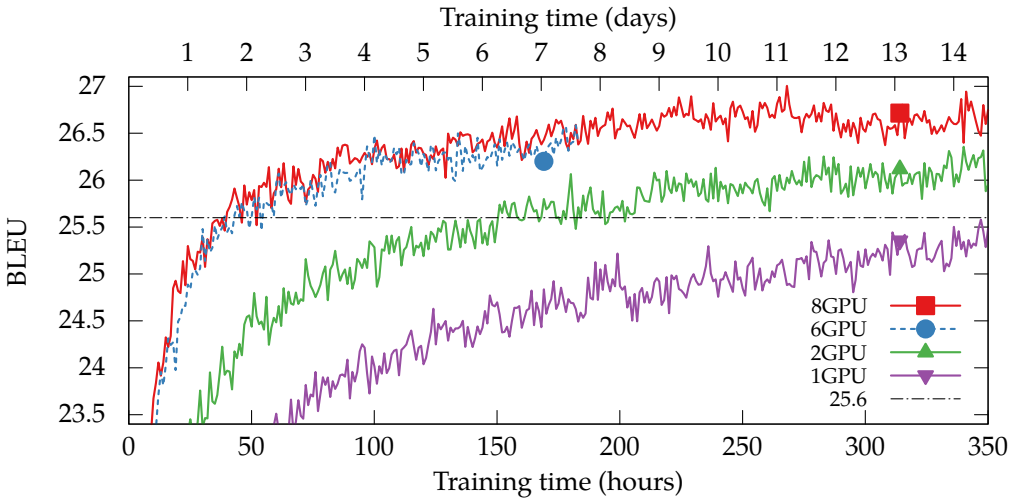

Figure 9: Effect of the number of GPUs. BLEU=25.6 is marked with a black line.

We can see that *two GPUs are more than three times faster than a single GPU* when measuring the Time Till Score and need much less training examples (i.e. they have lower Examples Till Score). Similarly, *eight GPUs are more than five times faster than two GPUs* and 1.7 times less training data is needed.

Recall that in Figure 6 we have shown that increasing the batch size from 1450 to 2000 has almost no effect on the BLEU curve. However, when increasing the effective batch size by using more GPUs, the improvement is higher than could be expected from the higher throughput.[28] We find this quite surprising, especially considering the fact that we have not tuned the learning rate and warmup steps (see the next section).

Tips on the Number of GPUs

- For the fastest BLEU convergence *use as many GPUs as available* (in our experiments up to 8).
- This holds *even when there are more experiments* to be done. For example, it is better to run one 8-GPUs experiment after another, rather than running two 4-GPUs experiments in parallel or eight single-GPU experiments in parallel.

[28] It would be interesting to try simulating multi-GPU training on a single GPU, simply by doing the update once after N batches (and summing the gradients). This is similar to the *ghost batches* of Hoffer et al. (2017), but using ghost batch size higher than the actual batch size. We leave this for future work.

4.8. Learning Rate and Warmup Steps on Multiple GPUs

4.8.1. Related Work

There is a growing number of papers on scaling deep learning to multiple machines with synchronous SGD (or its variants) by increasing the effective batch size. We will focus mostly on the question how to adapt the learning rate schedule, when scaling from one GPU (or any device, in general) to k GPUs.

Krizhevsky (2014) says "*Theory suggests that when multiplying the batch size by* k*, one should multiply the learning rate by* $\sqrt{k}$ *to keep the variance in the gradient expectation constant*", without actually explaining which theory suggests so. However, in the experimental part he reports that what worked the best, was a *linear scaling heuristics*, i.e. multiplying the learning rate by k, again without any explanation nor details on the difference between $\sqrt{k}$ scaling and k scaling.

The linear scaling heuristics become popular, leading to good scaling results in practice (Goyal et al., 2017; Smith et al., 2017) and also theoretical explanations (Bottou et al., 2016; Smith and Le, 2017; Jastrzebski et al., 2017). Smith and Le (2017) interpret SGD (and its variants) as a stochastic differential equation and show that the *gradient noise scale* $g = \epsilon\left(\frac{N}{B} - 1\right)$, where ϵ is the learning rate, N is the training set size, and B is the effective batch size. This noise "*drives SGD away from sharp minima, and therefore there is an optimal batch size which maximizes the test set accuracy*". In other words for keeping the optimal level of gradient noise (which leads to "flat minima" that generalize well), we need to scale the learning rate linearly when increasing the effective batch size.

However, Hoffer et al. (2017) suggest to use $\sqrt{k}$ scaling instead of the linear scaling and provide both theoretical and empirical support for this claim. They show that $\mathrm{cov}(\Delta w, \Delta w) \propto \frac{\epsilon^2}{NB}$, thus if we want to keep the the covariance matrix of the parameters update step Δw in the same range for any effective batch size B, we need to scale the learning rate proportionally to the square root of B. They found that $\sqrt{k}$ scaling works better than linear scaling on CIFAR10.[29] You et al. (2017) confirm linear scaling does not perform well on ImageNet and suggest to use Layer-wise Adaptive Rate Scaling.

We can see that large-batch training is still an open research question. Most of the papers cited above have experimental support only from the image recognition tasks (usually ImageNet) and convolutional networks (e.g. ResNet), so it is not clear whether their suggestions can be applied also on sequence-to-sequence tasks (NMT) with self-attentional networks (Transformer). There are several other differences as well: Modern convolutional networks are usually trained with *batch normalization*

[29] To close the gap between small-batch training and large-batch training, Hoffer et al. (2017) introduce (in addition to $\sqrt{k}$ scaling) so-called *ghost batch normalization* and *adapted training regime*, which means decaying the learning rate after a given number of steps instead of epochs.

(Ioffe and Szegedy, 2015), which seems to be important for the scaling, while Transformer uses *layer normalization* (Lei Ba et al., 2016).[30] Also, Transformer uses Adam together with an inverse-square-root learning-rate decay, while most ImageNet papers use SGD with momentum and piecewise-constant learning-rate decay.

4.8.2. Our Experiments

We decided to find out empirically the optimal learning rate for training on 8 GPUs. Increasing the learning rate from 0.20 to 0.30 resulted in diverged training (BLEU dropped to almost 0 after two hours of training). Similarly to our single-GPU experiments (Section 4.6), we were able prevent the divergence by increasing the warmup steps or by introducing gradient clipping (e.g. with `clip_grad_norm=1.0`, we were able to use learning rate 0.40, but increasing it further to 0.60 led to divergence anyway). However, *none of these experiments led to any improvements over the default learning rate* – all had about the same BLEU curve after few hours of training.

Jastrzebski et al. (2017) shows that "*the invariance under simultaneous rescaling of learning rate and batch size breaks down if the learning rate gets too large or the batch size gets too small*". A similar observation was reported e.g. by Bottou et al. (2016). Thus our initial hypothesis was that 0.20 (or 0.25) is the maximal learning rate suitable for stable training in our experiments even when we scale from a single GPU to 8 GPUs. Considering this initial hypothesis, we were surprised that we were able to achieve so good Time Till Score with 8 GPUs (more than 8 times smaller relative to a single GPU, as reported in Table 6). To answer this riddle we need to understand how learning rate schedules are implemented in T2T.

4.8.3. Parametrization of Learning Rate Schedules in T2T

In most works on learning rate schedules[31] the "time" parameter is actually interpreted as the number of epochs or training examples. For example a popular setup for piecewise-constant decay in ImageNet training (e.g. Goyal et al., 2017) is to divide the learning rate by a factor of 10 at the 30-th, 60-th, and 80-th epoch.

However, in T2T, it is the `global_step` variable that is used as the "time" parameter. So when increasing the effective batch size 8 times, e.g. by using 8 GPUs instead of a single GPU, the actual learning rate[32] achieves a given value after the same number of

[30] Applying batch normalization on RNN is difficult. Transformer does not use RNN, but still we were not successful in switching to batch normalization (and possibly ghost batch normalization) due to NaN loss errors.

[31] Examples of learning rate schedules are inverse-square-root decay, inverse-time decay, exponential decay, piecewise-constant decay, see `https://www.tensorflow.org/api_guides/python/train#Decaying_the_learning_rate` for TF implementations.

[32] By *actual* learning rate we mean the learning rate after applying the decay schedule. The `learning_rate` parameter stays the same in this case.

steps, but this means after 8 times less training examples. For the inverse-square-root decay, we have $actual\_lr(steps) = c \cdot steps^{-0.5} = \frac{1}{\sqrt{8}} \cdot actual\_lr(steps \cdot 8)$, where c is a constant containing also the `learning_rate` parameter. So with 8 GPUs, if we divide the `learning_rate` parameter by $\sqrt{8}$, we achieve the same actual learning rate after a given number of training examples as in the original single-GPU setting.

This explains the riddle from the previous section. *By keeping the `learning_rate` parameter the same when scaling to* k *times bigger effective batch, we actually increase the actual learning rate* $\sqrt{k}$ *times*, in accordance with the suggestion of Hoffer et al. (2017).[33] This holds only for the `linear_warmup_rsqrt_decay` (aka `noam`) schedule and ignoring the warmup steps.

If we want to keep the same learning rate also in the warmup phase, we would need to divide the warmup steps by k. However, this means that the maximum actual learning rate will be $\sqrt{k}$ times higher, relative to the single-GPU maximal actual learning rate and this leads to divergence in our experiments. In deed, many researchers (e.g. Goyal et al., 2017) suggest to use a warmup when scaling to more GPUs in order to prevent divergence. Transformer uses learning rate warmup by default even for single-GPU training (cf. Section 4.6), but it makes sense to use more warmup training examples in multi-GPU setting.

In our experiments with 8 GPUs and the default learning rate 0.20, using 8k warmup steps instead of the default 16k had no effect on the BLEU curve (it was a bit higher in the first few hours, but the same afterwards). Further decreasing the warmup steps resulted in a retarded BLEU curve (for 6k) or a complete divergence (for 2k).

Tips on Learning Rate and Warmup Steps on Multiple GPUs

- Keep the `learning_rate` parameter at its optimal value found in single-GPU experiments.
- You can try decreasing the warmup steps, but less than linearly and you should not expect to improve the final BLEU this way.

4.9. Resumed Training

T2T allows to resume training from a checkpoint, simply by pointing the `output_dir` parameter to a directory with an existing checkpoint (specified in the `checkpoint` file). This may be useful when the training fails (e.g. because of hardware error), when we need to continue training on a different machine or during hyper-parameter search, when we want to continue with the most promising setups. T2T saves also Adam

[33] In addition to suggesting the $\sqrt{k}$ learning-rate scaling, Hoffer et al. (2017) show that to fully close the "generalization gap", we need to train longer because the absolute number of steps (updates) matters. So from this point of view, using steps instead of epochs as the time parameter for learning rate schedules may not be a completely wrong idea.

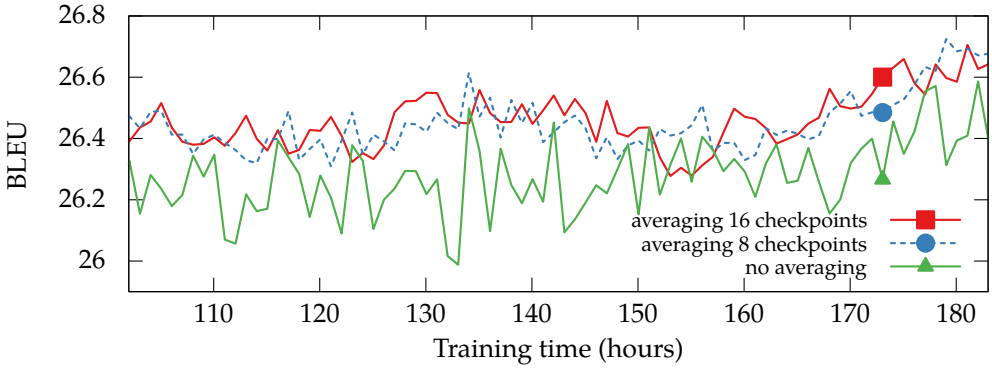

Figure 10: Effect of checkpoint averaging. All trained on 6 GPUs.

momentum into the checkpoint, so the training continues almost as if it had not been stopped. However, it does not store the position in the training data – it starts from a random position. Also the relative time (and wall-clock time) in TensorBoard graphs will be influenced by the stopping.

Resumed training can also be exploited for changing some hyper-parameters, which cannot be meta-parametrized by the number of steps. For example, Smith et al. (2017) suggest to increase the effective batch size (and number of GPUs) during training, instead of decaying the learning rate.

Yet another usage is to do domain adaptation by switching from (large) general-domain training data to (small) target-domain training data for the few last epochs. In this case, consider editing also the learning rate or learning rate schedule (or faking the `global_step` stored in the checkpoint) to make sure the learning rate is not too small.

4.10. Checkpoint Averaging

Vaswani et al. (2017) suggest to average the last 20 checkpoints saved in 10-minute intervals (using `utils/avg_checkpoints.py`). According to our experiments slightly better results are achieved with averaging checkpoints saved in 1-hour intervals. This has also the advantage that less time is spent with checkpoint saving, so the training is faster.

Figure 10 shows the effect of averaging is twofold: the averaged curve has lower variance (flickering) from checkpoint to checkpoint and it is almost always better than the baseline without averaging (usually by about 0.2 BLEU). In some setups, we have seen improvements due to averaging over 1 BLEU. In the early phases of training, while the (baseline) learning curve grows fast, it is better to use fewer checkpoints for

	Manual		Automatic Scores				
#	Ave %	Ave z	BLEU	TER	CharacTER	BEER	System
–	–	–	**23.8**	**0.662**	**0.582**	**0.543**	T2T 8 GPUs 8 days
1	**62.0**	**0.308**	*22.8*	*0.667*	*0.588*	*0.540*	uedin-nmt
2	*59.7*	*0.240*	20.1	0.703	0.612	0.519	online-B
3	55.9	0.111	20.2	0.696	0.607	0.524	limsi-factored
	55.2	0.102	20.0	0.699	-	-	LIUM-FNMT
	55.2	0.090	20.2	0.701	*0.605*	0.522	LIUM-NMT
	54.1	0.050	20.5	0.696	0.624	0.523	CU-Chimera
	53.3	0.029	16.6	0.743	0.637	0.503	online-A
8	41.9	-0.327	16.2	0.757	0.697	0.485	PJATK

Table 7: WMT17 systems for English-to-Czech and our best T2T training run. Manual scores are from the official WMT17 ranking. Automatic metrics were provided by `http://matrix.statmt.org/`. For *TER metrics, lower is better. Best results in bold, second-best in italics.

averaging. In later phases (as shown in Figure 10, after 4.5–7.5 days of training), it seems that 16 checkpoints (covering last 16 hours) give slightly better results on average than 8 checkpoints, but we have not done any proper evaluation for significance (using paired bootstrap testing for each hour and then summarizing the results).

The fact that resumed training starts from a random position in the training data (cf. Section 4.9) can be actually exploited for "forking" a training to get two (or more) copies of the model, which are trained for the same number of steps, but independently in the later stages and thus ending with different weights saved in the final checkpoint. These semi-independent models can be averaged in the same way as checkpoints from the same run, as described above. Our preliminary results show this helps a bit (on top of checkpoint averaging).

Tips on Checkpoint Averaging

- Use it. Averaging 8 checkpoints takes about 5 minutes, so it is a "BLEU boost for free" (compared with the time needed for the whole training).
- See the tools for automatic checkpoint averaging and evaluation described in Section 2.4.

5. Comparison with WMT17 Systems

Table 7 provides the results of WMT17 English-to-Czech news translation task, with our best Transformer model (BIG trained on 8 GPUs for 8 days, averaging 8 checkpoints) evaluated using the exact same implementation of automatic metrics. While the automatic evaluation is not fully reliable (see e.g. the high BLEU score for CU-Chimera despite its lower manual rank), we see that the Transformer model out-

performs the best system in BLEU, TER, CharacTER and BEER, despite it does not use any back-translated data, reranking with other models (e.g. right-to-left reranking) nor ensembling (as is the case of uedin-nmt and other systems). Note that our Transformer uses a subset of the constrained training data for WMT17, so the results are comparable.

6. Conclusion

We presented a broad range of basic experiments with the Transformer model (Vaswani et al., 2017) for English-to-Czech neural machine translation. While we limit our exploration to the more or less basic parameter settings, we believe this report can be useful for other researchers. In sum, experiments done for this article took about 4 years of GPU time.

Among other practical observations, we've seen that for the Transformer model, larger batch sizes lead not only to faster training but more importantly better translation quality. Given at least a day and a 11GB GPU for training, the larger setup (BIG) should be always preferred. The Transformer model and its implementation in Tensor2Tensor is also best fit for "intense training": using as many GPUs as possible and running experiments one after another should be preferred over running several single-GPU experiments concurrently.

The best performing model we obtained on 8 GPUs trained for 8 days has outperformed the WMT17 winner in a number of automatic metrics.

Acknowledgements

This research was supported by the grants 18-24210S of the Czech Science Foundation, H2020-ICT-2014-1-645452 (QT21) of the EU, SVV 260 453, and using language resources distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (LM2015071).

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of ICLR*, 2015.

Bojar, Ondřej, Zdeněk Žabokrtský, Ondřej Dušek, Petra Galuščáková, Martin Majliš, David Mareček, Jiří Maršík, Michal Novák, Martin Popel, and Aleš Tamchyna. The Joy of Parallelism with CzEng 1.0. In *Proceedings of the Eighth International Language Resources and Evaluation Conference (LREC'12)*, pages 3921–3928, Istanbul, Turkey, May 2012. ELRA, European Language Resources Association. ISBN 978-2-9517408-7-7.

Bojar, Ondřej, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In Sojka, Petr, Aleš Horák, Ivan Kopeček, and Karel Pala,

editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Artificial Intelligence, pages 231–238. Masaryk University, Springer International Publishing, 2016. ISBN 978-3-319-45509-9.

Bojar, Ondřej, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. Findings of the 2017 Conference on Machine Translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark, September 2017a. ACL.

Bojar, Ondřej, Yvette Graham, and Amir Kamran. Results of the WMT17 Metrics Shared Task. In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark, September 2017b. ACL.

Bottou, Léon. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_25. URL https://doi.org/10.1007/978-3-642-35289-8_25.

Bottou, L., F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *ArXiv e-prints*, June 2016. URL https://arxiv.org/abs/1606.04838.

Cettolo, Mauro, Marcello Federico, Luisa Bentivogli, Jan Niehues, Sebastian Stüker, Katsuhito Sudoh, Koichiro Yoshino, and Christian Federmann. Overview of the IWSLT 2017 Evaluation Campaign. In *Proceedings of the 14th International Workshop on Spoken Language Translation (IWSLT)*, pages 2–14, Tokyo, Japan, 2017.

Goyal, Priya, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR*, 2017. URL http://arxiv.org/abs/1706.02677.

Hoffer, Elad, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In Guyon, I., U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1731–1741. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6770-train-longer-generalize-better-closing-the-generalization-gap-in-large-batch-training-of-neural-networks.pdf.

Ioffe, Sergey and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Jastrzebski, Stanislaw, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J. Storkey. Three Factors Influencing Minima in SGD. *CoRR*, abs/1711.04623, 2017. URL http://arxiv.org/abs/1711.04623.

Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *Proceedings of ICLR*, 2017. URL http://arxiv.org/abs/1609.04836.

Krizhevsky, Alex. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014. URL http://arxiv.org/abs/1404.5997.

Lee, Jason, Kyunghyun Cho, and Thomas Hofmann. Fully Character-Level Neural Machine Translation without Explicit Segmentation. *CoRR*, 2016. URL http://arxiv.org/abs/1610.03017.

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

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of ACL 2002*, pages 311–318, Philadelphia, Pennsylvania, 2002.

Popović, Maja. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. ACL. URL http://aclweb.org/anthology/W15-3049.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of ACL 2016*, pages 1715–1725, Berlin, Germany, August 2016. ACL. URL http://www.aclweb.org/anthology/P16-1162.

Shazeer, N. and M. Stern. Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. *ArXiv e-prints*, Apr. 2018. URL https://arxiv.org/abs/1804.04235.

Smith, Samuel L. and Quoc V. Le. A Bayesian Perspective on Generalization and Stochastic Gradient Descent. In *Proceedings of Second workshop on Bayesian Deep Learning (NIPS 2017)*, Long Beach, CA, USA, 2017. URL http://arxiv.org/abs/1710.06451.

Smith, Samuel L., Pieter-Jan Kindermans, and Quoc V. Le. Don't Decay the Learning Rate, Increase the Batch Size. *CoRR*, 2017. URL http://arxiv.org/abs/1711.00489.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Guyon, I., U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6000–6010. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

You, Yang, Igor Gitman, and Boris Ginsburg. Scaling SGD Batch Size to 32K for ImageNet Training. *CoRR*, abs/1708.03888, 2017. URL http://arxiv.org/abs/1708.03888.

Address for correspondence:
Martin Popel
popel@ufal.mff.cuni.cz
Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics, Charles University
Malostranské náměstí 25, 118 00 Praha 1
Czech Republic