OpenNMT System Description for WNMT 2018
800 words/sec on a single CPU

Jean Senellart, Dakun Zhang, Bo Wang,
Guillaume Klein, J.P. Ramatchandirin, Josep Crego
SYSTRAN Research
Paris
first.last@systrangroup.com

Alexander M. Rush
School of Engineering
and Applied Sciences
Harvard University
srush@seas.harvard.edu

Abstract

We present a system description of the OpenNMT neural machine translation (NMT) entry for the WNMT 2018 evaluation. In this work, we developed a heavily optimized NMT inference model targeting a high-performance CPU system. The system uses a combination of four techniques, all of them lead to significant speed-ups in combination: (a) sequence distillation, (b) architecture modifications, (c) precomputation, particularly of vocabulary, and (d) CPU targeted quantization. This work achieves the fastest performance of the shared task, and led to development of new features that have been integrated to OpenNMT and available to the community.

1 Introduction

As neural machine translation becomes more widely deployed in production environments, it becomes also increasingly important to serve translations models in as fast and as memory-efficient way possible, both on dedicated GPU and on standard CPU hardwares. The WNMT 2018 shared task[1] focused on comparing different systems on both accuracy and computational efficiency (Birch et al., 2018).

This paper describes the entry for the OpenNMT system to this competition. Our specific interest was to explore the different techniques for training and optimizing CPU models for very high throughput while preserving highest possible accuracy compared to state-of-the-art. While we did not put real focus on memory and docker size footprint, we applied basic optimization techniques to reduce the final size of our models.

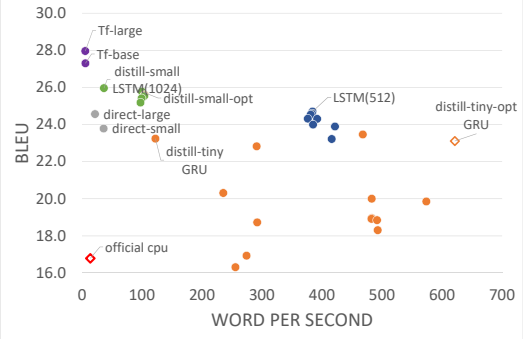

Figure 1: Pareto on accuracy and throughput. The different models on the plot are described in the paper, the corresponding BLEU score calculated on *newstest2014*, and throughput in word per second measured on non dedicated M5 instances.

Our strategy for the shared task was to take advantage of four main optimization techniques: (a) sequence-level *distillation*, in particular cross-class distillation from a transformer model (Vaswani et al., 2017) to an RNN, (b) *architecture search*, changing the structure of the network by increasing the size of more efficient modules, reducing the size of the more costly and replacing default gated units, (c) specialized *precomputation* such as reducing dynamically the runtime target vocabulary (Shi and Knight, 2017), and (d) *quantization* and faster matrix operations, based on the work of Devlin (2017) and `gemmlowp`[2]. All of these methods are employed in a special-purpose C++-based decoder *CTranslate*[3]. The complete training workflow including data preparation and distillation is described in Section 2. Inference techniques and quantization are described in Section 3.

Our experiments compare the different approaches in terms of speed and accuracy. A meta

[1] https://sites.google.com/site/wnmt18/shared-task

[2] https://github.com/google/gemmlowp

[3] https://github.com/OpenNMT/CTranslate

question of this work is to decide which models represent interesting and useful points on the Pareto curve. The main results are described in Figure 1. From these results we highlight two models which were submitted to the shared task: Our first system `distill-small-opt` is a 2-layer LSTM network that is only -2.19 BLEU points behind that reference transformer model, but with a speed-up of x18. Our second system achieves 23.11 on WNMT 2018 English-German *newstest2014* (-4.85 behind the reference model) but with an additional decoding speed-up of x8. The final model reaches 800 words/sec on the dedicated evaluation CPU hardware and is the fastest CPU model submitted.

We additionally report several new results about recent work in NMT and model efficiency: (a) we show that distillation of transformer model outperforms training result of a strong RNN model - extending findings of Crego and Senellart (2016) who reported that student models could outperform their teacher for reference RNN-based model. (b) we show that distillation from a transformer neural network to a simple RNN neural network is efficient, (c) we compare quantitatively different quantizations, and (b) we give an improved algorithm to dynamically select target vocabulary for a given batch. Finally, we also report several complementary experiments that resulted in systems inside of the pareto convex border. For instance, we compare using 8-bit quantization to 16-bit quantization.

2 Training and Distillation

Following the shared task setup, we use the data set provided by WNMT 2018, which is a preprocessed and tokenized version of WMT 2014 on English-German translation[4]. The training data contains about 4.5M sentence pairs. We use *newstest2013* as the validation set and *newstest2014* and *newstest2015* as the testing sets. Before training, we trained a 32K joint byte-pair encoding (BPE) to preprocess the data (Sennrich et al., 2015). Hence, the generated subword vocabulary is less than 40K word pieces. We limit the sentence length to 100 based on BPE preprocessing in both source and target side (excluding only 0.31% of the training corpus). After decoding, we remove the BPE joiners and evaluate the tokenized output with multi-bleu.perl (Koehn et al., 2007).

2.1 Teacher Model: Transformer

Transformer networks (Vaswani et al., 2017) are the current state-of-the art in many machine translation tasks (Shaw et al., 2018). The network directly models the representations of each sentence with a self-attention mechanism. Hence much longer term dependencies can be learned, which is especially important for language pairs like English-German. In addition, transformer allows to easily parallelise the MLE training process across multiple GPUs. However, a large number of parameters are needed by the network to obtain its best performance. In order to reduce the model size, we applied knowledge distillation, a technique that has proven successful for reducing the size of neural models. We considered the transformer network as our teacher network.

We used *OpenNMT-tf*[5] to train two transformer based systems: base and large described in Table 2 with their evaluation in Table 3. For both, the learning rate is set to 2.0 and warmup steps 8000, we average the last 8 checkpoints to get the final model. Our baseline system outperforms the provided baseline Sockeye model by +0.37 BLEU on *newstest2014*.

2.2 Distillation to RNN

To train our smaller student system, we follow the sequence-level knowledge distillation approach described by Kim and Rush (2016). First, we build the full transformer as above. Next, we use the teacher system to retranslate all the training source sentences to generate a simplified target sentences set. Then, we use this simplified corpus (original source and newly generated target) to train a student system. The student system can be assigned with smaller network size, in our case an RNN-based sequence-to-sequence models similar to Bahdanau et al. (2014)

Results from Crego and Senellart (2016) show that the distillation process not only improves throughput of the student models and reduce their size, but can also improve the translation quality if the width of the network is not reduced too much. Also, these distilled systems have the interesting feature that they performed almost identically when reducing beam search size K from $K = 5$ to $K = 2$. Seemingly the smaller model

[4] https://nlp.stanford.edu/projects/nmt/

[5] https://github.com/OpenNMT/OpenNMT-tf

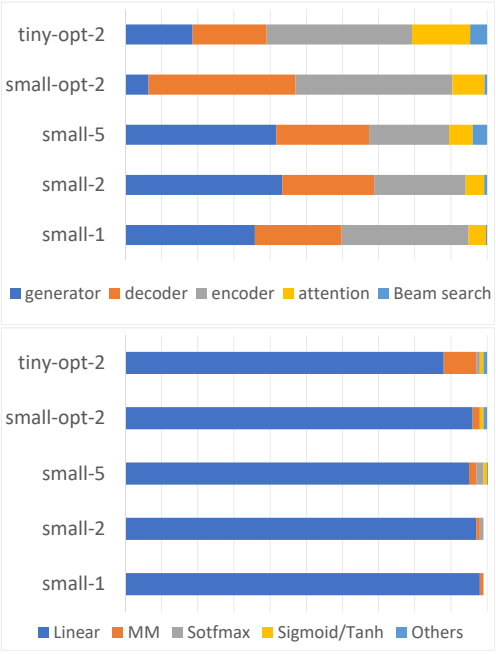

Figure 2: Profiling of the throughput during inference on *newstest2014*. Each line is a different system. The `small-1,2,5` are distilled systems with respective beam size of 1, 2, 5. `distill-tiny-opt` and `distill-small-opt` are the final systems. The total decoding time for each of these systems (in seconds) are from top to bottom: 60, 464, 754, 662 and 629. The top table show the distribution of time per component of the network, while the bottom table show the distribution of time for network operators.

learns from more consistent translations and can produce effective results with simpler syntactic structure and even word choices. One major question here was to find out whether cross-class distillation works as well, here from an transformer model to student RNN model.

Our student NMT system follows the standard architecture of Luong et al. (2015). It is implemented as an encoder-decoder network with multiple layers of a RNN with Long Short-Term Memory hidden (Hochreiter and Schmidhuber, 1997) units and attentional architecture. Full details of the *OpenNMT*[6] system are given in Klein et al. (2017).

2.3 Model Analysis

Profiling of the throughput of our student system, before and after further optimizations, is presented in Figure 2. Here we compare along two splits, the components of the system: encoder, decoder, attention, word generation, and beam search; and model components: linear, MM (matrix multiply, primarily in attention), softmax, activations, and others. From this analysis we gather the following facts:

- The most costly part of the inference is the generator, that is the final Linear layer feeding Softmax with weights corresponding to each single vocab of the target language. This is by far the largest matrix multiplication of the system: $(B, W) * (W, V)$ (with B being the batch*beam size, W the width of the network, and V the size of the vocabulary).
- Although the cost of the encoder for beam size 1 is higher than the decoder, the decoder cost - including generator and attention model - grows linearly with the beam size.
- The most costly operation are the Linear matrix-vector multiplies in the RNNs and generator, contributing more 95% of the complete processing time.

For these observations, it is obvious that we need to optimize the efficiency of linear matrix-vector operation, reduce the amount required, and have a special focus on the generation layer.

These observations led us to several further model experiments. First we tried different model combination based on a "fat encoder, thin decoder" spirit: ie. increasing if necessary the number of operations in the encoder part if we can in parallel reduce the number of operations in the decoder. Second, we experimented with replacing LSTM cell with GRU cells (Cho et al., 2014). Indeed LSTM cells have 4 gates while GRU cells only 3, so potentially leading to a 25% improvement in speed with similar performance can be reached. Note though, that we found GRU requires more care in optimization while a naive SGD optimization is generally sufficient for LSTM-RNN training.

3 Inference Optimizations

3.1 Implementation: CTranslate

We begin with a custom RNN decoder. CTranslate is the open-source inference engine for OpenNMT models (designed for Lua Torch). The code is implemented in raw C++ with minimal dependencies using the popular Eigen linear algebra library. CTranslate's goal is to offer a lightweight and embeddable solution for executing models. It benefits from Eigen efficiency and is about 20% faster

[6] https://github.com/OpenNMT/OpenNMT

than a Torch application using Intel MKL. Additionally, the use of C++ over a garbage-collected language ensures a predictable and reduced memory usage. All additional optimizations are built on top of this library.

3.2 Generator Vocabulary Reduction

As seen in the last section, the word generation matrix operations and softmax require significant time. This time is scaling linearly with the effective target language vocabulary, which starts at 40K.

There has been significant work in reducing this cost. We start with the word alignment method, presented in Shi and Knight (2017), which uses alignments computed on the training corpus and for each sentence, it selects target words aligned to each source word. Hence building a reduced vocabulary of target words to be used when translating a source sentence.

To increase the coverage of the selected meanings without increasing the size of the mapping model, we first extract words in the target language that are unaligned - e.g. determiners when translating from English to French. We kept the 100 most frequent such words and call them 0-gram meanings. Then we go through 1-gram, 2-gram, ..., n-gram sequences of words in the source sentence. For each source sequence, we consider its N-best translation hypotheses. All target words present in such N-best hypotheses are kept in the target vocabulary. To account for translation hypotheses we use a phrase table extracted from word alignments.

The method extends single word vocabulary mappings to multi-word expressions[7]. We have multiple criterion to extract a vocabulary map: the maximal sequence size n, the maximum number N of translation hypotheses, the minimum frequency of a phrase pair. The efficiency of a vocabulary map can be evaluated through the coverage of the predicted meanings for a reference test set, the final translation quality and the average number of meaning per vocab, and the actual time spent in generator layer (Linear and Softmax).

Table 1 compares several vocabulary maps with these different metrics. Compared to Shi and Knight (2017), our approach with multiple n-gram length phrase enables better match-rate than the 1-gram approach (saturating the Test Coverage (TC) at 80%).

3.3 Quantization

Another important area of optimization is the cost of linear matrix-operations. To speed these up, we use 16-bit signed integer quantization method proposed in Devlin (2017). To further optimize on the AWS M5 instance used in the competition and powered with INTEL Skylake processors, we extend the approach to AVX2 and AVX512 SIMD instruction sets. Switching from SSE4 to AVX2, then from AVX2 to AVX512 instructions set gave additional speed boost of respectively $+12\%$ and $+6\%$.

3.4 Other Experiments

We explored several other methods which largely resulted in negative results but could be interesting for other contexts.

Decrease the sentence length: For BPE preprocessing, we try using 64K merges which can generate shorter sentences. The average sentence length for 32K BPE is about 29.1 and for 64K BPE, it is about 27.7 tokens. Assuming the same efficiency in the vocabulary mapping, increasing the size of the vocabulary could therefore have gain about 5% additional speed-up just by the reduction of the sentence length. However, the tradeoff here was not clearly a win.

8-bit quantization: To reduce further the system size, we also considered use of `gemmlowp`[8]. `gemmlowp` is a library allowing quantization to unsigned 8 bits integer through dynamic offset/multiplier/shift parameters. Like our implementation of 16-bit quantization, the low precision is only for storing the parameters, the dot product is using larger register for accumulating intermediate result of the operation. `gemmlowp` usage is for embedded application where speed but also power usage is critical. The idea was tempting, but it was not clear if such quantization schema could actually outperform quantization using SIMD extended instruction set on modern processors. We ran comparative tests using AVX2 instructions set and found out that for multiplications of large matrixes $(20, 1024) * (1024, 512)$ - optimized IN-

[7] For instance `speed test` translated by `test de vitesse` in French is covered by 0-grams $\varnothing \rightarrow$`de`, and 1-grams `speed`$\rightarrow$`vitesse`, `test`$\rightarrow$`test`. However, `once more` translated by `à nouveau` will need the 2 additional meanings `à` and `nouveau` that are only covered when using 2-gram meanings.

[8] https://github.com/google/gemmlowp

Max Sequence	Min Freq	# Meaning	vocab per token	Test Coverage	Linear Time [s]	SoftMax Time [s]	File Size	BLEU
					143.86	4.97	239M	23.24
1	1	50	2	30%	3.78	0.11	295M	21.98
2	1	100	24	86%	5.97	0.16	902M	23.13
2	1	150	25	86%	6.63	0.18	918M	23.16
2	1	50	22	85%	5.34	0.14	846M	23.09
2	2	100	22	85%	4.08	0.11	324M	23.02
2	2	150	22	85%	3.95	0.11	324M	23.02
2	2	50	20	84%	3.95	0.11	322M	23.03
3	1	50	30	90%	5.23	0.15	1127M	23.16

Table 1: Evaluations of n-gram vocabulary mappings on newstest2014.

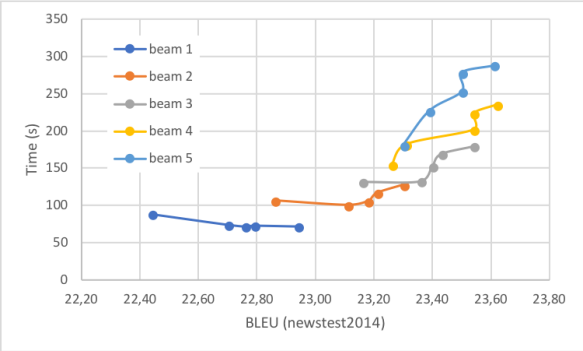

Figure 3: Evaluations on different beam size and batch size

T16 implementation was about 3 times faster than `gemmlowp` UINT8 implementation[9]. Main reason being that AVX2 (and AVX512 twice faster) have a very powerful multiply and add instructions[10] allowing to perform in one single cycle the dot product of vectors 32*INT16 and at the same time, the pair accumulation in a vector 16*INT32.

4 Results

After tuning, we settled on two optimal NMT systems based on the distilled training data. Table 2 lists the different configurations for these two systems:

- `distill-small`, uses a bidirectional RNN with 2 LSTM layers with each hidden layer having 1024 nodes. We use a word embedding size of 512 and set the dropout to 0.3. The batch size is set to 64 and the default learning rate is 1.0 with sgd optimization.
- `distill-tiny`, uses a smaller network, with GRU layers, 512 hidden size. On the encoder side, we have 2 layers, while on the decoder side, only 1 layer is set. We use Adam optimization with the starting learning rate 0.0002.

Both systems are trained up to 10 epochs.

Table 3 shows our internal evaluations. For system `distill-small-opt`, the CPU time during decoding improves from 1694.16 seconds to 621.17 seconds (saving 63.3%), with a loss of only 0.19 BLEU score, on *newstest2014*. For system `distill-tiny-opt`, the trends are similar. 80.3% cpu time is saving, while only 0.13 BLEU score is lost.

We also compare the influence of quantization hyperparameters, e.g. vmap and quantize model, on tiny distilled RNN (`distill-tiny`). Quantize runtime and vmap both can save the decoding time about 50%. The quantized model also halves the model size.

4.1 Beam size and Batch size

We further test the impact on the decoding performance (BLEU) and CPU time of different beam size and batch size on system `distill-tiny-opt`. Figure 3 shows for a fixed batch size, when we increase the beam size from 1 to 3, the accuracy increases as well. While for beam size 3, 4 and 5, there is no significant difference in accuracy which is consistent with previous findings on distilled system. Interestingly, for a fixed beam size, we notice also a slight improvement of accuracy when increasing the batch size. This is a side-effect of the dynamic vocabulary mapping per batch.

These experiments also show that the decoding cost increases with larger beam and batch size. For beam size $K = 1$ (in blue), we process more sentences inside each batch and the CPU cost reduces along the increasing of batch size. While for the others, larger beam size and larger batch size both cost more computational effort.

[9] On Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz

[10] `_mm256_madd_epi16` and `_mm512_madd_epi16`

Transformer	Large	N=6, d=512, d_{ff}=4096, h=8
	Base	N=6, d=512, d_{ff}=2048, h=8
direct RNN	Large	b-LSTM, 4 layers*1024, embed=512, optim=sgd
	Small	b-LSTM, 2 layers*1024, embed=512, optim=sgd
distilled RNN	Small	b-LSTM, 2 layers*1024, embed=512, optim=sgd
	Tiny	GRU, enc:2, dec:1 *512, embed=256, optim=adam

Table 2: Configurations for the different systems presented in the paper

		quantize runtime	vmap	quantize model	*newstest2014*		*newstest2015*		model size
					cpu time [s]	BLEU	cpu time [s]	BLEU	
Transformer	Large	-	-	-	11279.97	27.96	8339.10	29.95	1.4G
	Base	-	-	-	10795.16	27.30	7511.08	29.36	1.2G
direct RNN	Large	-	-	-	2859.57	24.56	2073.90	27.24	618M
	Small	-	-	-	1713.54	23.78	1313.39	26.37	416M
distilled RNN	Small	-	-	-	1694.16	25.96	1300.24	28.62	416M
	Small-opt	Y	Y	Y	621.17	25.77	478.08	28.60	207M
	Tiny	-	-	-	506.67	23.24	384.76	26.09	141M
	Tiny	Y	-	-	286.89	23.24	219.85	26.03	141M
	Tiny	Y	Y	-	105.80	23.18	77.10	25.95	141M
	Tiny-opt	Y	Y	Y	99.81	23.11	77.76	25.75	72M

Table 3: Evaluations on NMT systems (the suffix "-opt" means it is the final submission)

As a result, we choose beam $K = 2$ and batch 2, balancing the performance and computation cost.

4.2 Docker Image Size

We did not invest effort into reduce image size during the preparation of the submission. Our fastest system has a docker image of 200M for an effective 72Mb size for the model and less than 15Mb for additional code and resources. Post-submission, we looked at reducing this 110Mb overhead coming from operating system and misc tools. Without huge effort - we managed to reduced this overhead to 70Mb, and it is probably easy to reduce even further.

4.3 Other Engineering Considerations

We note that at this level of optimization, especially the use of quantization, the speed measurement is very dependent on low level memory management mechanisms[11], and therefore on other processes running on the same instance, especially due to the critical importance of the L3/L4-cache shared between the different cores. In particular, we observed that 4 parallel processes using fastest model were only reaching a x3 speed boost. To go further on parallel decoding, one would need to implement different ad-hoc mechanisms such as:

- synchronization points between the parallel decoders to avoid waste of memory cache transfer
- grouping of the sentences by sentence-length to optimize CPU usage on the different cores

All the optimizations performed for that submission were focussed on the Linear layers - on the final fastest submitted system, the profile of Figure 2 shows emerging room for optimization: even though the Linear share remains preponderant, some potential additional gain (between 5 and 10%) could be achieved by focusing on the other operators.

5 Conclusion

This work presents the OpenNMT submission to the 2018 Shared Task of WNMT. We show that training with distillation using an optimized RNN sequence-to-sequence system we can produce a very competitive model for CPU demonstrating again the powerful effect of the distillation process and for the first time its application cross-class (Transformer→RNN). This positive result implies that text simplification through distillation could be applied to more contexts.

Even though our submission was dedicated to a specific RNN-based network, most of the presented optimizations, aiming by different means to reduce and optimize the matrix multiplication operations can apply for other types of architectures.

Our final system does show an impressive increase in speed of 110x compared to the baseline system and achieves a throughput of 800 word/s

[11] This effect was actually observed during the preparation of the system: a same test could benchmark with a fluctuation of about 10-15% depending on the time of the day, and probable load of the shared server hosting the virtual instance.

on a single core which is the fastest reported so far.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Alexandra Birch, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Yusuke Oda. 2018. Findings of the second workshop on neural machine translation and generation. In *The Second Workshop on Neural Machine Translation and Generation*.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Josep Maria Crego and Jean Senellart. 2016. Neural machine translation from simplified translations. *CoRR*, abs/1612.06139.

Jacob Devlin. 2017. Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the CPU. *CoRR*, abs/1705.01991.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Y Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. *CoRR*, abs/1606.07947.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Accepted to ACL 2017 Conference Demo Papers*, Vancouver, Canada. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *CoRR*, abs/1803.02155.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking runtime vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 574–579.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.