# Fat Encoder, Thin Decoder
# OpenNMT System Description for WNMT 2018

**Anonymous ACL submission**


## Abstract

In this paper, we present OpenNMT sequence to sequence neural machine translation (NMT) system for the WNMT 2018 evaluation optimized for CPU inference. This work led to the development of new features that have been integrated to OpenNMT and already available to the community.


## 1 Introduction

In this paper, we describe the systems we built and selected for WNMT 2018[1]. Our specific interest was to explore the different techniques for training and optimizing CPU models with high throughput while keeping high accuracy. Even though 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 (delivered as fully functional dockers).

For this purpose, we explored the following techniques:

- Model distillation as introduced by Kim and Rush (2016) to transfer knowledge from a strong baseline neural network (the teacher) to a smaller network (the student); we were in particular interested to explore the possibility of distilling knowledge from a transformer model (Vaswani et al., 2017) to a light-weight RNN model,
- 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,
- Using C++ based decoder *CTranslate*[2],

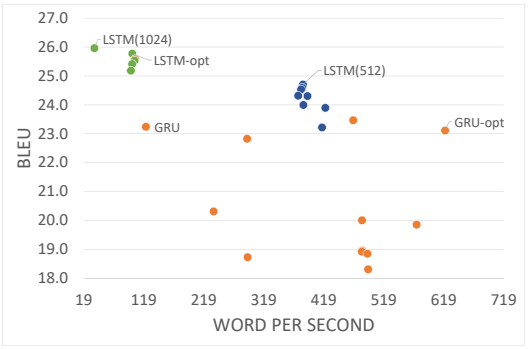


Figure 1: Pareto on accuracy and throughput.

- Reducing dynamically the runtime target vocabulary extending (Shi and Knight, 2017) ideas,
- Quantizing parameters of the models as presented in Devlin (2017) or `gemmlowp`[3]

The complete training workflow is therefore based on the following steps:

- Preparation of data and sub-tokenization model - described in section 2.1.
- Training of strong a Teacher transformer model, and translation of the full dataset with this model - described in section 2.2.
- Training of student model on translated data - described in section 2.3 and 2.4.
- Quantizing the generated model - described in section 3.3.

By combining all these techniques, we generated a large number of models and picked two of them presenting interesting trade-off between speed and quality and on the frontier of the Pareto curve (see Figure 1). Our first system `LSTM-opt-2` is a simple 2-layer LSTM network

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

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

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

is only -2.19 points behind reference transformer model, 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 x6 at about 1000 words/sec on the evaluation hardware.

Also, 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 systems could outperform their teacher for reference RNN-based model.

Finally, we also report several other negative experiments that produced systems inside of the pareto convex border. For instance, we compared using 8-bit quantization to 16-bit quantization.

Beyond the presented systems, this paper brings the following contributions: a/ we prove that distillation from a transformer neural network to a simple RNN neural network is efficient, b/ we compare quantitatively different quantizations, c/ we improve algorithm to select dynamically target vocabulary for a given batch.

## 2 Training

### 2.1 Data

We use constrained 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. And 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[5].

### 2.2 Transformer

We train a transformer based system (Vaswani et al., 2017) as our teacher system. Compared with RNN/CNN models, transformer based model does not rely on recurrent or convolution network. It directly models the representations of each sentence with self-attention mechanism. Hence much longer information in the sentences can be learned especially important for language pairs like English-German and it currently achieves state-of-art results for machine translation (Shaw et al., 2018)

Another advantage for transformer based system is that the training process can be parallelized easily. However, transformer networks still require a large number of parameters and also large memory to achieve the best performance. As a result, the parallelizable training, the flexibility in modeling both long-range/local dependencies, and the state-of-the-art performance make this architecture suitable to act as the teacher system.

We used *OpenNMT-tf*[6] to train two transformer based systems: base and large described in Table 1 with their evaluation. 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 in particular the provided baseline sockeye model with +0.37 BLEU on newstest 2014.

### 2.3 Distillation

We follow the method described in Kim and Rush (2016). Firstly, we build the full transformer based MT system (Vaswani et al., 2017) as the teacher system. Secondly, we use the teacher system to translate all the 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, in our case a sequence-to-sequence models similar to Bahdanau et al. (2014). The student system can be assigned with smaller network size and this process is called distillation.

According to Crego and Senellart (2016), distillation process not only improves throughput of the 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 for beam 2 or beam 5 beam search. The interpretation for these facts being that the teacher produced simplified translations. That is, translations that are usually closer, in terms of syntactic structure and even word choices, to the input sentences than refer-

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

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

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

| | $N$ | $d_{model}$ | $d_{ff}$ | $h$ | newstest2014 | newstest2015 |
|---|---|---|---|---|---|---|
| Transformer (base) | 6 | 512 | 2048 | 8 | 27.30 | 29.36 |
| Transformer (large) | 6 | 512 | 4096 | 8 | 27.96 | 29.95 |

Table 1: Evaluation on transformer based teacher system

ence translations. Thus the student model better manage to learn and generalize this simplified knowledge. One major question here was to find out whether distillation from an transformer model could benefit a student RNN model. We proved it was the case, since the best distilled system outperforms a larger model trained with the original data.

### 2.4 Sequence to Sequence

Our baseline NMT system follows the architecture presented in Luong et al. (2015). It is implemented as an encoder-decoder network with multiple layers of a RNN with Long Short-Term Memory hidden units (Zaremba et al., 2014) and attentional architecture from Luong et al. (2015) available on *OpenNMT*[7]. Full details of the system are given in Klein et al. (2017).

Profiling of the throughput of such system is presented in Figure 2. 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.
- Transversally to the different component of the network, the most costly module is the Linear module cumulating more 95% of the complete processing time

For these observations, it is obvious that we needed to optimize the efficiency of Linear operation (basic matrix multiplication), reduce their number and have a special focus on the generation layer. It was also interesting to note that focus on any other operator was useless.

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

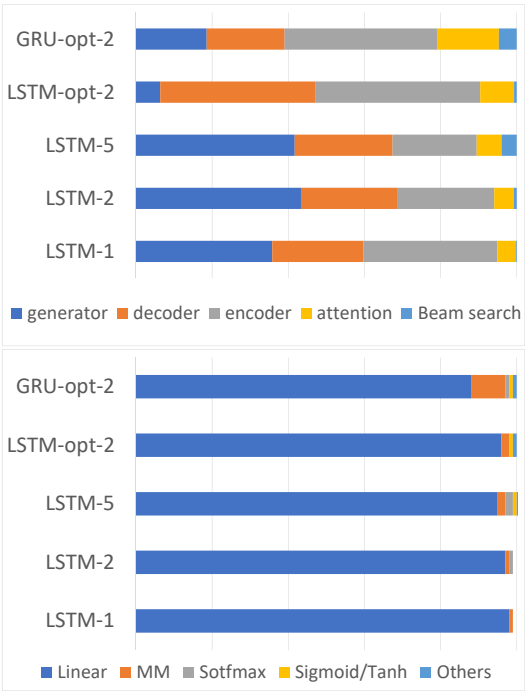


Figure 2: Profiling of the throughput during inference of newstest2014. Each line is a different system. The `LSTM-1,2,5` are baseline systems with respective beam size of 1, 2, 5. `GRU-opt-2` and `LSTM-opt-2` are the presented systems. The total decoding time for each of these systems (in seconds) are 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.

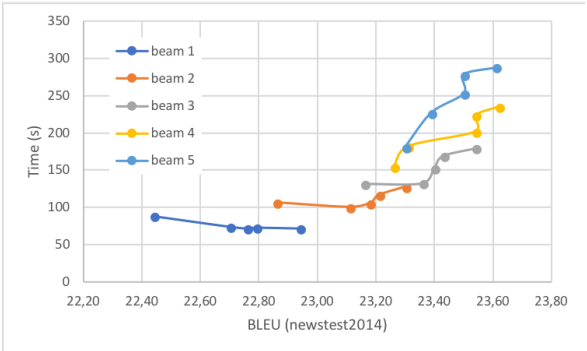


Figure 3: Evaluations on different beam size and batch size

On the model architecture, 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.

Also, we tried to substitute LSTM (Hochreiter and Schmidhuber, 1997) 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. We are not aware of any comprehensive study comparing relative efficiency of GRU vs LSTM. Our findings show that almost similar performance can be reached. However, GRU requires more care in optimization method while a naive SGD optimization is generally sufficient for LSTM-RNN trainings.

## 3 Decoding optimization

### 3.1 C++ Decoder: CTranslate

CTranslate is an open source inference engine for OpenNMT models (designed for LuaTorch version) that uses raw C++ with minimal dependencies. The project can load trained models, read the computation graph, and run inference using the popular Eigen library.

CTranslate's goal is to offer a lightweight and embeddable solution for executing models and to support advanced CPU optimization such as quantization and parallel translation. Without these optimization, it still benefits from Eigen efficiency and is about 20% faster than an Torch application using Intel MKL. Additionally, the use of C++ over a garbage-collected language ensures a predictable and reduced memory usage.

### 3.2 Reduction of vocabulary size

To reduce the size of the vocabulary for each batch, we started up from word alignment method presented in Shi and Knight (2017). This approach is essentially a 1-gram approach. However to increase the coverage of the selected meanings without increasing the size of the mapping model, we first extracted words in the target languages that are spontaneously generated (generally without alignment with the source) - 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 trough 1-gram, 2-gram, ..., n-grams. For each source phrase $f_1 ... f_n$, we consider the target phrase as a bag of words, and we check the meanings of the covered 0, ... $n-1$-grams and discard these meanings from the bag of words. If words remain after examining all of the sub-sequence, we consider these words as new meaning for the current phrase. This approach extends vocabulary mapping to multi-word expressions[8].

We have multiple criterion to extract a vocabulary map through the maximal sequence (MS), the maximal frequency of the pair (MF), the maximum number of meaning (KM). The efficiency of a vocabulary map can be evaluated through the coverage of the predicted meanings for a reference test set (TC), the final translation quality and the average number of meaning per vocab, and the actual time spend in generator layer (Linear and Softmax). We present in Table 2 several vocabulary maps with these different metrics and in Table 4 the impact of using such a vocabulary map on the presented systems. Compared to Shi and Knight (2017), our approach with multiple $n-gram$ length enables better match-rate than the 1-gram approach (saturating the coverage TC at 80%).

### 3.3 Quantization

We use 16-bit signed integer quantization method proposed in Devlin (2017) - however to better optimize speed on AWS M5 instance, supporting AVX512 SIMD instructions, we extended the approach to AVX2 and AVX512. Switching from SSE4 to AVX2, then from AVX2 to AVX512 instructions set gave additional speed boost of respectively $+12\%$ and $+6\%$.

### 3.4 Further potential optimizations and negative results

- Decrease the sentence length: For BPE preprocessing, we try using 64K merges which can generate much 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.
- 8-bit quantization: To reduce further the

[8] For instance `speed test` translated by `test de vitesse` in French is covered by 0-grams, and 1-gram. However, `once more` translated by `nouveau` will need 2 additional meanings not covered in 0-gram or 1-gram meanings

| MS | MF | KM | vocab per token | TC | 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 2: Evaluations of n-gram vocabulary mappings on newstest2014.

system size, we also considered use of `gemmlowp`[9]. `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 INT16 implementation was about 3 times faster than `gemmlowp` UINT8 implementation[10]. Main reason being that AVX2 (and AVX512 twice faster) have a very powerful multiply and add instructions[11] 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.

- Other architecture: In the same spirit of reducing the decoder layer, Devlin (2017) introduces an hybrid decoder with only one RNN attentional layer followed by fully connected layers with residual connections. The author reports reaching same quality than a multiple layers RNN with attention. We tried several variants around this idea but could not reproduce the same results.

  We also tried to reduce the width of the RNN more, but impact on quality was too important.

## 4 Evaluations

In this section, we show our internal evaluation results. We train two NMT systems based on the synthetic training data with *OpenNMT*, a Lua-Torch version implementation on sequence to sequence MT. Table 3 lists the different configurations for these two systems.

For system `LSTM-opt-2`, we use 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. For system `GRU-opt-2`, we train with 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.

Evaluations are shown in Table 4. For system `LSTM-opt-2`, 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 `GRU-opt-2`, the trends are similar. 80.3% cpu time is saving, while only 0.13 BLEU score is lost.

We also compare the different influence with the options quantize runtime, vmap and quantize model, on system `GRU-opt-2`. The option quantize runtime and vmap both can save the decoding time about 50%. The option quantize model functions well on halving the model size.

[9] https://github.com/google/gemmlowp
[10] On Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz
[11] `_mm256_madd_epi16` and `_mm512_madd_epi16`

| | RNN type | RNN size | encoder layers | decoder layers | embedding size | optim |
|---|---|---|---|---|---|---|
| LSTM | b-LSTM | 1024 | 2 | 2 | 512 | sgd |
| GRU | GRU | 512 | 2 | 1 | 256 | adam |

Table 3: Configurations for the two student systems

| | quantize runtime | vmap | quantize model | newstest2014 | | newstest2015 | | model size |
|---|---|---|---|---|---|---|---|---|
| | | | | cpu time [s] | BLEU | cpu time [s] | BLEU | |
| LSTM | - | - | - | 1694.16 | 25.96 | 1300.24 | 28.62 | 416M |
| LSTM-opt | Y | Y | Y | 621.17 | 25.77 | 478.08 | 28.60 | 207M |
| GRU | - | - | - | 506.67 | 23.24 | 384.76 | 26.09 | 141M |
| GRU | Y | - | - | 286.89 | 23.24 | 219.85 | 26.03 | 141M |
| GRU | Y | Y | - | 105.80 | 23.18 | 77.10 | 25.95 | 141M |
| GRU-opt | Y | Y | Y | 99.81 | 23.11 | 77.76 | 25.75 | 72M |

Table 4: Evaluations on two student systems (the suffix "-opt" means it is the final submission)

### 4.1 Beam size and Batch size

We further make a full testing on the decoding performance (BLEU) and CPU time of different beam size and batch size on system `GRU-opt-2`. As shown in Figure 3, 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 as a side-effect of the dynamic vocabulary mapping and showing that some additional optimizations could be obtained on vocabulary mapping selection.

At the same time, the decoding cost is increasing when larger beam and batch size are setting (Figure 3). For beam size 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.

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

## 5 Docker size

We did not focus on docker image size during the preparation of the submission, and our fastest system docker is 200M for effective 72Mb size for the model and less than 15Mb for additional code and resources. Post-submission, we looked at reducing this 100Mb overhead coming from operating system and found that easy optimizations could be applied - for instance like remove useless dependencies like Python. Without huge effort - we managed to reduced this overhead to 70Mb and it is probably easy to reduce even more.

## 6 Further work and Conclusion

We train through distillation a fat encoder, thin decoder sequence to sequence NMT system for WNMT 2018 running on one single CPU. The models use quantization leading to both a size reduction and a speed gain. Several additional optimizations could have been applied, such as full graph quantization, augmentation of vocabulary size, more improvement on vocabulary mapping...

We need to note that at this level of optimization, especially the use of quantization, makes the speed measurement very dependent 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 process using fastest model were only reaching a x3 speed boost. To go further on parallel decoding, one would need to implement synchronization points between parallel decoding to avoid waste of memory cache transfer.

## References

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

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.

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.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.