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

Anonymous ACL submission

Abstract

This paper presents the OpenNMT 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

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

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 with high throughput while maintaining high accuracy, although allowing for a small decrease below the 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.

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) *quantization* and faster matrix operations, based on the work of Devlin (2017) and `gemmlowp`[2] and (d) specialized *precomputation* such as reducing dynamically the runtime target vocabulary (Shi and Knight, 2017). 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.

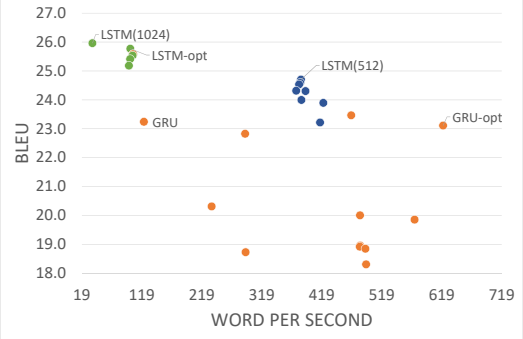

Figure 1: Pareto on accuracy and throughput.

Our experiments compare the different approaches in terms of speed and accuracy. A meta 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 `LSTM-opt-2` 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 x108.

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

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

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

Trans.	N	d	d_{ff}	h	N2014	N2015
Base	6	512	2048	8	27.30	29.36
Large	6	512	4096	8	27.96	29.95

Table 1: Evaluation of base and large transformer teacher model on Newstest 2014 and 2015.

The final model reaches 1000 words/sec on the evaluation CPU hardware.

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 systems 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 other negative experiments that result 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[5].

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

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

2.1 Teacher Model: Transformer

For our base teacher model, we train a transformer NMT system (Vaswani et al., 2017). Unlike standard sequence-to-sequence models, the transformer does not use recurrent or convolution networks. It directly models the representations of each sentence with self-attention mechanism. Hence much longer term dependencies can be learned, which is especially important for language pairs like English-German. It currently achieves state-of-art results for machine translation (Shaw et al., 2018) Another advantage for transformer based system is that the MLE training process can be parallelized easily across multiple GPUs.

Due to their current effectiveness, it would be tempting to use the transformer architecture directly for our final system (and other submissions took this approach). However we decided not to use this strategy. For one transformer networks still require a large number of parameters and also large memory to achieve the best performance. Additionally some of the training benefits of transformer are less apparent in auto-regressive inference.

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 our 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 the provided baseline Sockeye model by +0.37 BLEU on newstest 2014.

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

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

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 learns from "smoother" 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*[7] 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, actiations, and other. 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 is the Linear matrix-vector multiplies in the RNNs and generator, contributing more 95% of the complete processing time

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

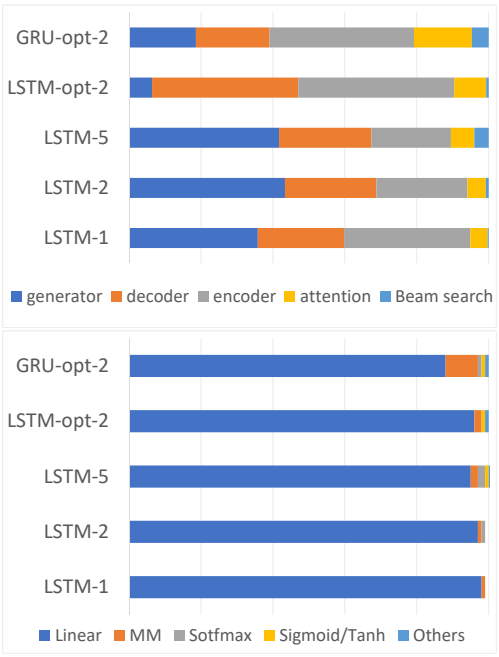

Figure 2: Profiling of the throughput during inference on newstest2014. Each line is a different system. The `LSTM-1,2,5` are baseline systems with respective beam size of 1, 2, 5. `GRU-opt-2` and `LSTM-opt-2` are the final systems. The total decoding time for each of these systems (in seconds) are from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

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 than an 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 scales linearly with the effective target language vocabulary, which starts at 32K.

There has been significant work in reducing this cost. We start with the word alignment method, presented in Shi and Knight (2017), which selects target words based on unigram alignments. To increase the coverage of the selected meanings without increasing the size of the mapping model, we first extract 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 through unigram, bigram, $ldots$, n-grams alignments. 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 keep these as possible targets. 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, the maximal frequency of the pair, the maximum number of meaning. 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 2 compares several vocabulary maps with these different metrics. 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

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 better optimize speed on the AWS M5 instance used in the competition, which supports AVX512 SIMD instructions, we extend 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 Negative Inference Results

We explored several other methods which largely resulted in 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. However, the tradeoff here was not clearly a win.

8-bit quantization: To reduce further the 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

[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

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

Max Sequence	Max 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 2: Evaluations of n-gram vocabulary mappings on newstest2014.

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.

Non-recurrent architectures: 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 Results

After tuning, we settled on two optimal NMT systems based on the distilled training data. Table 3 lists the different configurations for these two systems:

- `LSTM-opt-2`, 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.
- `GRU-opt-2`, 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.

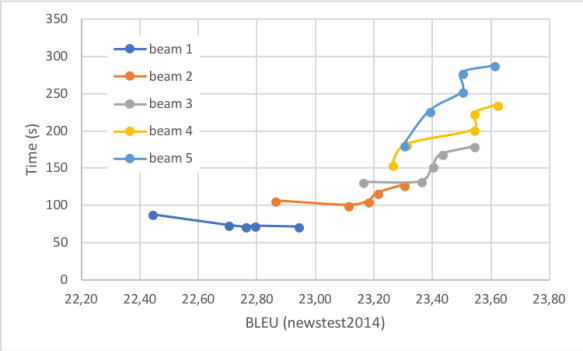

Figure 3: Evaluations on different beam size and batch size

Both systems are trained up to 10 epochs.

Table 4 shows our internal evaluations. 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 influence of quantization hyperparameters, e.g. vmap and quantize model, on system `GRU-opt-2`. Quantize runtime and vmap both can save the decoding time about 50%. The quantized model also halves the model size.

[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 test the impact on the decoding performance (BLEU) and CPU time of different beam size and batch size on system `GRU-opt-2`. 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. This shows that some additional optimizations could be obtained on vocabulary mapping selection taking batching into account.

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.

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

4.2 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 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 further.

4.3 Other Considerations

We 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.

5 Conclusion

This work presents the OpenNMT submission to the 2018 Shared Task of WNMT focusing on efficient and NMT. We show that training with distillation using an optimized "fat encoder, thin decoder" RNN sequence-to-sequence system we can produce a very efficient CPU based model. The models additionally uses quantization leading to both a size reduction and a speed gain. Several further optimizations could be applied, such as full graph quantization, augmentation of vocabulary size, more improvement on vocabulary mapping.

Our final system though does show a significant increase in speed of todo...

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.