

OpenNMT: Open-Source Toolkit for Neural Machine Translation

Guillaume Klein[†], Yoon Kim^{*}, Yuntian Deng^{*}, Jean Senellart[†], Alexander M. Rush^{*}
Harvard University^{*}, Systran[†]

Abstract

We describe an open-source toolkit for neural machine translation that supports research development of attention-based encoder-decoder models. The toolkit prioritizes efficiency, modularity, and extensibility to make it reasonable for researchers to experiment with variants of neural machine translation that explore different feature representations, model architectures, and source modalities, while maintaining competitive performance and tractable training requirements. The toolkit consists of modeling and decoding support, as well as detailed pedagogical documentation about the underlying methodologies.

1 Introduction

Neural machine translation (NMT) is a new methodology for machine translation that has led to remarkable improvements particularly in terms of human judgment of translation quality compared to rule-based and statistical machine translation systems (SMT). Originally developed using pure sequence-to-sequence models (Seq2Seq) and improved upon using attention-based variants (Seq2Seq+Attn), it has now become a standard methodology for machine translation, as well as an effective approach for other related NLP tasks such as dialogue, parsing, and summarization.

As neural machine translation becomes standardized, it becomes more important for the machine translation and NLP community to develop standard reference implementations for researchers to benchmark against, learn from, and extend upon. Just as the SMT community benefited greatly from toolkits like Moses (Koehn, 2007) for phrase-based MT and the CDec toolkit (Ding et al., 2015) for

Figure 1: Schematic view of neural machine translation. The **red** source words are first mapped to word vectors and then fed into a recurrent neural network (RNN). Upon seeing the $\langle \text{eos} \rangle$ symbol, the final time step initializes a target **blue** RNN. At each target time step, *attention* is applied over the source RNN and combined with the current hidden state to produce a prediction $p(w_t | w_{1:t-1}, x)$ of the next word. This prediction is then fed back into the target RNN.

syntax-based MT, standard NMT toolkits can provide the foundation to compare results and develop a more robust open-source community. A toolkit should aim to provide a shared frameworks for developing and comparing open-source SMT systems that are complete and flexible enough for research development, while at the same time being efficient and accurate enough to be used production contexts.

Currently there are many different existing NMT implementations. Many systems such as those developed in industry by Google (Schroeder et al., 2015), Microsoft, and Baidu, are closed source, and are unlikely to be released with unrestricted licenses. Many other systems such as GroundHog, Blocks, tensorflow-seq2seq, and our own seq2seq-attn, exist mostly as research code. These libraries provide partial functionality and minimal support. All provide some subset of efficiency, completeness, high accuracy, modularity, and clear documentation. Perhaps most promising is the Edinburgh NeMaTus system originally based on NYU’s system. NeMaTus provides high-accuracy translation, many options, clear documentation, and has been used in several successful research projects. We hope to complement this type of system to provide a useful open-source NMT framework for the NLP community in academia and industry.

With this goal in mind, we introduce *OpenNMT* (<http://opennmt.net>), an open-source framework for neural machine translation. OpenNMT is a complete NMT implementation. In

addition to providing code for the core translation tasks, OpenNMT was designed with three criteria:

1. prioritize first training and test efficiency
2. maintain model modularity and readability
3. support significant research extensibility

This technical report describes the how the first-release of the system targets these criteria. We begin by briefly surveying the background for NMT, describing the high-level implementation details, and then describing specific case studies for the three criteria. We end by showing preliminary benchmarks of the system in terms of accuracy, speed, and memory usage for several translation and translation-like tasks.

2 Background

Neural machine translation has now been extensively described in many excellent papers and tutorials (see for instance <https://sites.google.com/site/acl16nmt/home>). We therefore give only a condensed overview here.

NMT takes a conditional language modeling view of translation (as opposed to the noisy channel view of SMT). Formally NMT models the probability of a target sentence $w_{1:T}$ given a source sentence $x_{1:S}$ as $p(w_{1:T}|x) = \prod_{t=1}^T p(w_t|w_{1:t-1}, x; \theta)$. This distribution is estimated using an attention-based encoder-decoder architecture (?). A source encoder recurrent neural network (RNN) maps each source word to a word vector, and processes these to a sequence of hidden vectors $\mathbf{h}_1, \dots, \mathbf{h}_S$. The target decoder combines an RNN hidden representation of previously generated words (w_1, \dots, w_{t-1}) with source hidden vectors to predict scores for each possible next word. A softmax layer is then used to produce a distribution $p(w_t|w_{1:t-1}, x; \theta)$. Crucially the source hidden vectors are processed through an attention pooling layer that weights each source word relative to its expected contribution to the target prediction. The complete model is trained end-to-end to minimize the negative log-likelihood of the training corpus. An unfolded network diagram is shown in Figure 1.

In practice, there are also several other important details that contribute to model effectiveness: (a) It is important to use a gated RNN such as an LSTM (?) or GRU (?) which help the model learn

long-term features. (b) Translation requires relatively large, stacked RNNs, which consist of several layers (2-16) of RNN at each time step (?). (c) Input feeding, where the previous attention vector is fed back into the input as well as the predicted word, has been shown to be quite helpful for machine translation (?). (d) Test-time decoding is done through *beam search* where multiple hypothesis target predictions are considered at each time step.

3 Implementation

OpenNMT is a complete system and library for learning, training, and deploying neural machine translation models. The system is successor to the *seq2seq-attn* system developed at Harvard, using roughly the same external interface, but with a complete rewrite of the internals to ease readability and generalizability. It includes vanilla NMT models along with support for attention, gating, stacking, input feeding, regularization, beam search and all other options necessary for state-of-the-art performance.

The system is implemented using the Torch mathematical framework and neural network library, and can be easily be extended using Torch's internal standard neural network components. The current version uses its own RNN implementation for fine-grained control of memory allocation.

The system has been developed completely in the open on GitHub at (<http://github.com/opennmt/opennmt>) and is MIT licensed. The first version has primarily (intercontinental) contributions from Systran Paris and the Harvard NLP group. Since official beta release, the project has been starred by over 500 users, and there have been active development by those outside of these two organizations. The project has an active forum for community feedback.

One nice aspect of NMT as a model is its relative compactness. The complete OpenNMT system including preprocessing is roughly 4K lines of code. For comparison the Moses SMT framework including language modeling is over 100K lines. This makes the system easy to completely understand for newcomers and contributors. The project is fully autonomous including also a simple language independent reversible tokenization and detokenization tools.

4 Design Goals

As the the low-level details of NMT have been covered in previous works, we focus this tech report on the top-down design goals of OpenNMT. We focused particularly on three ordered criteria: system efficiency, code modularity, and model extensibility. Here we present case studies of progress on each goal.

4.1 System Efficiency

As NMT systems can take from days to weeks to train, training efficiency is a paramount research concern, and the top priority of OpenNMT. Slightly faster training can make be the difference between plausible and impossible experiments.

Optimization: Memory Sharing With few exceptions, GPUs are currently limited to 12 GB of memory. When training NMT models, the memory size limits the plausible batch size of the system, and thus directly impacts training time of the system. Neural network toolkits, such as Torch, are often designed to trade-off extra memory allocations for speed and declarative simplicity. For OpenNMT, we wanted to have it both ways, and so we implemented an external memory sharing system that exploits the known time-series control flow of NMT systems and aggressively shares the internal buffers between clones. The possible sharing is dynamically calculated by exploration of the network graph before starting the training. This makes the system slightly less flexible than toolkits such as Element-RNN (?), but provides a saving of almost 70% of GPU memory. This in turn allows for much larger batch sizes.

Optimization: Multi-GPU OpenNMT additionally supports multi-GPU training using data parallelism. The implementation is relatively straightforward, each GPU has a replica of the master parameters and process independent batches during training phase. Two modes are available: synchronous and asynchronous training

- in synchronous training, batches on parallel GPU are run simultaneously and gradients aggregated to update master parameters before synchronization on each GPU for the following batch
- in asynchronous training, batches are run independent on each GPU, and independent gradients accumulated to the master copy

of the parameters. Asynchronous SGD is known to provide better convergence.

The parallel implementation uses low level optimized primitive for multi-GPU communication¹.

Case Study: C/Mobile/GPU Decoders During training, NMT systems require significant code complexity and storage to facilitate back-propagation-through-time and parameter updates. At test time the system is much less complex, and only requires (i) forwarding values through the network and (ii) running a beam search that is much simplified compared to SMT. To exploit this asymmetry, OpenNMT includes several different decoders specialized for different run-time environments: a batched GPU decoder for very quickly translating a large set of sentences, a simple single-instance decoder for use on mobile devices, and a specialized C decoder. The last decoder is particularly nice to have for industrial use as it can run on CPU in standard production environments. The decoder reads the structure of the network from Lua and then uses the Eigen package to implement the basic linear algebra necessary for decoding. Decoders are all available at the OpenNMT GitHub (<http://github.com/opennmt>).

4.2 Modularity for Research

While training efficiency was a primary concern, the secondary goal was a desire for code readability at the advanced undergraduate-level. We targeted this goal by explicitly separating out the above optimizations from the core model, and by documenting each module with mathematical diagrams describing how it connects to the underlying neural network descriptions. To test whether this approach would allow novel feature development we experimented with two case studies.

Case Study: Factored Neural Translation In feature-based or factored neural translation models (?), instead of simply generating a word at each time step, the model generates both word and its features. For instance, the model might have a separate case feature, in which case it would model the probability of the lower-cased word form and the case marker. This extension requires modifying both the output of the decoder to generate multiple symbols, and also the input to the decoder to take in a word and its features. In Open-

¹see NCCL - <https://github.com/NVIDIA/nccl>

NMT both of these aspects are abstracted from the core translation code, and therefore we were able to add factored translation by modifying the input network to instead process the feature-based representation, and the output generator network to instead produce multiple conditionally independent predictions. This option can be turned on by modifying the training data to include the factored words.

Case Study: Attention Networks The use of attention over the encoder at each step of translation is crucial for the model to perform well. The default method is to utilize the gloabl attention mechanism proposed by (). However there are many other times of attention that have recently proposed including local attention (?), sparse-max attention (?), Hierarchical attention (?) among others. As this is simply a module in OpenNMT it can easily be substituted. Recently the Harvard NLP group developed a new method known as structured attention, that utilizes graphical model inference to compute this attention. The method was quite involved and required custom CUDA code to compute efficiently. However the method is modularized to fit the Torch NN interface and can therefore be directly used in OpenNMT to substitute for standard attention.

Finally, we wanted the project not to depend on third party tools like commonly used Moses tokenizer (in perl) and BPE implementation (in python). Moses tokenizer integrates language specific tokenization heuristic that are not necessary for RNN-based approach. So we introduced a simple tokenization schema - called "reversible tokenization" with following characteristics:

- the tokenization includes markers allowing the detokenization so that all language knowledge is part of the model
- the tokenization rules are extremely simple and come in 2 modes. In aggressive mode, transition between letter and number or separator, between two separators, or between a number and a separator is corresponding to a tokenization mark. In conservative mode, numbers, letters, and symbols `[-_]` are grouped together, and the same for numbers and number separator symbol `[, ,]`.

Also tokenizer also perform BPE splitting (?).

Example of tokenized sentences are given in table `reftab:token`.

4.3 Extensibility

The last goal of OpenNMT is to realize the deep learning is a very quick moving area and that likely within the next couple years there will be many unexpected applications of these style of methods. Already we see related, but very different styles of work, e.g. in variational seq2seq variation auto-encoders (?), one-shot learning (?), or memory network (?) based models.

Case Study: Image-to-Text As an extensibility case study of the library, we experimented with implementing a complete attention-based image-to-text translation system (?) using the OpenNMT library. Qualitatively this is a very different problem than standard machine translation as the source sentence is an image. However, the future of translation may require this style of (multi-)modal inputs (e.g. <http://www.statmt.org/wmt16/multimodal-task.html>). In particular, we adapted the `im2markup` system (?) to instead use OpenNMT as a library. Instead of using word embeddings, this model requires a deep convolution over the source input. However as this part of the network is pluggable, it could be naturally defined in Torch. In fact, excepting preprocessing, the entire adaptation requires only 500 lines of code and is open sourced as github.com/opennmt/im2text

5 Benchmarks

In this section we document preliminary runs of the model. We expect performance and memory usage to improve with further development.

These benchmarks are run using a machine Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz, 256GB Mem, trained on 1 GPU TITAN X (Pascal version) with CUDA v. 8.0. Run on English-to-German (EN→DE) and German-to-English (DE→EN) using the WMT 2015² dataset. These benchmarks are run using the default parameters of OpenNMT which is a two-layer encoder-decoder LSTM model with 500 hidden units per layer. For comparison we also run the NeMaTus (<https://github.com/rsennrich/nematus>) system on the same data.

Results are show in Table 5. The ...

Additionally we also trained OpenNMT on several non-standard translation tasks. First is a sum-

²<http://statmt.org/wmt15>

| Language | Sent/Sec | Memory | BLEU |
|----------|----------|--------|-------|
| DE→EN | 216.7 | 2.5 GB | 17.02 |
| EN→DE | 211.3 | 2.6 GB | 20.67 |

Table 1: Performance Results. Several languages

Table 2: Speed Results. Multi-GPU, distillation, c decoder

marization model () ...

Finally we trained a multilingual translation model following ?). This model is a 5x5 translation model translating across romance language. It translates from and to French, Spanish, Portuguese, Italian, and Romanian (FR,ES,PT,IT,RO↔FR,ES,PT,IT,RO). Training data is 4M sentences and was selected from the open parallel corpus³ and specifically from Europarl, GlobalVoices and Ted. Corpus was selected to be multi-source, multi-target: each sentence has its translation in the 4 other languages. The motivation of this selection was to evaluate the inter-language learning and not the additional sentences from each language available in other language pairs. Corpus was tokenized using shared Byte Pair Encoding of 32k. Results are presented in 5 and we can see that each language pair quality receives a huge gain in the multi-way training. This is a clear evidence of the contribution from each language pair to the interlingua representation between all these very close languages.

Picture of demo application running

6 Conclusion

³<http://opus.lingfil.uu.se>

| | ES | FR | IT | PT | RO |
|----|----------------|---------------|---------------|---------------|---------------|
| ES | - | 32.71 (+5.43) | 28 (+4.64) | 34.41 (+6.08) | 28.73 (+6.45) |
| FR | 32.87 (+3.3) | - | 26.32 (+4.25) | 30.89 (+5.16) | 25.95 (+6.64) |
| IT | 31.64 (+5.34) | 31.03 (+5.81) | - | 27.96 (+4.98) | 24.27 (+5.9) |
| PT | 35.32 (+10.38) | 34.08 (+4.68) | 28.09 (+5.55) | - | |
| RO | | | | | |

Table 3: Performance Results for the twenty language pairs with the unique translation model. In parenthesis, score improvement compared to individual models trained with only the language pair data.

Table 4: Example of reversible tokenization