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 sequence-to-sequence models. The toolkit prioritizes simplicity, modularity, and efficiency to make it reasonable for researchers to experiment with variants of neural machine translation that explore different feature representations, model architectures, and source (multi)-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 shown remarkable gains particularly in terms of human evaluation. Originally popularized by the work of () and expanded upon by (). Over the last year the systems have been further refined and developed by work such as ().

In this work we describe a new open-source toolkit for developing neural machine translation systems, known as *OpenNMT*. The system is motivated by frameworks, such as Moses and CDec developed for statistical machine translation (SMT). These toolkits 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 several existing NMT systems. Including Google NMT (), ... Several of these systems are proprietary. Several others are mainly research system such as seq2seq-attn. Perhaps most similar to OpenNMT is the NeMaTus system (), based on the system of () which aims to provide the same type of feature set as OpenNMT, using a different underlying neural network toolkit.

In developing OpenNMT we prioritized three different factors, which are described in this technical report: (1) Our top priority was training and decoding speed. NMT systems are notoriously slow to train, often requiring weeks of training time on the latest GPU hardware and significant test resources. We targeted this issue by implementing multi-GPU training, using aggressive memory sharing, and developing a specialized CPU decoder. (2) Our second priority was system modularity and teachibility. We intend OpenNMT as a living research system, and so the codebase was developed to provide a self-documenting overview of NMT. We discuss how this approach allowed us to add factored models () to the system. (3) Finally NMT is a very quick moving research area, and we would like the system to support new research areas as they develop. To demonstrate this approach we abstracted out the core of OpenNMT as a library, and describe the case study of using OpenNMT for image-to-text translation.

This report describes the background for NMT, the main aspects of system development, and the preliminary results from using the system in practice.

2 Background

Neural machine translation has now been extensively described in many excellent papers and tutorials (), which we briefly summarize. NMT systems take a conditional language modeling view of translation (as opposed to the noisy channel view of SMT systems). This means

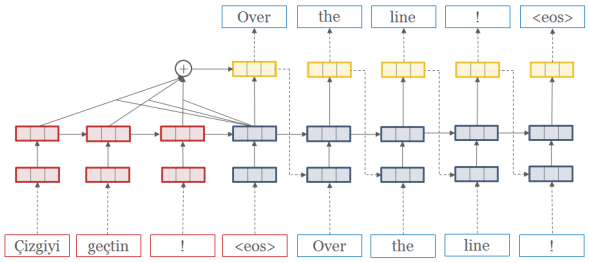

Figure 1: Schematic view of neural machine translation. The redred source words are first mapped to word vectors and then fed into a recurrent neural network (RNN). Upon seeing the ⟨eos⟩ symbol, the final time step initializes a target blueblue RNN. At each target time step, *attention* is applied over 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.

they model the probability of a target sentence $w_{1:T}$ given a source sentence x as $p(w_{1:T}|x) = \prod_1^T p(w_t|w_{1:t-1}, x; \theta)$. This distribution is modeled using an attention-based encoder-decoder architecture (). In particular a source encoder recurrent neural network (RNN) maps each source word to a word vector, and processes these to a sequence of hidden vectors $\mathbf{h}_1, \ldots, \mathbf{h}_S$. To produce the language model probability a target decoder RNN, combines a representation of previously generated words $(w_1, ...w_{t-1})$ with these source hidden vectors and a softmax layer to produce $p(w_t|w_{1:t-1}, x; \theta)$. The source hidden vectors are utilized through an attention layer, a type of pooling 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.

In practice, there are also several 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 (). OpenNMT provides command-line support for each of these options.

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.

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. Originally, external RNN components libraries were used, but the current version uses its own RNN implementation for fine-grained control of memory allocation. The training aspects of the library are highly tuned for single or multi-GPU usage. The test component can be run either on CPU or GPU, and uses batched beam search to speed up the translation process.

The system has been developed fully as an open-source project on GitHub at `http://github.com/opennmt/opennmt` with most contributions coming from Systran and, across the ocean, 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.

One nice aspect of the implementation 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.

4 Design Goals

As there are now many descriptions and details of low-level RNN and beam search implementations, we skip many of these details and instead focus on the main design desiderata that motivated OpenNMT: efficiency, modularity, and extensibility. We point interested readers to the code for implementation notes.

4.1 Efficiency Optimizations

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

Memory Sharing With some exception standard consumer 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. 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.

Multi-GPU OpenNMT additionally supports basic multi-GPU training. The implementation is relatively straightforward, each GPU runs its own instance of the model. We run async SGD

C/Mobile/GPU Decoders During training NMT systems require signficant code complexity and storage to implement backpropagation through time and parameter updates. At test time the system is much less complex, and only requires forwarding values through the network and running a beam search that is much simplified compared to SMT. To exploit this, OpenNMT includes several different decoders optimized for different 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 useful for industial 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.

4.2 Structural Modularity

While training efficiency was a primary concern, this goal was balanced with the desire to keep the code readable and modifiable by an advance undergraduate. 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 factored neural translation models (), instead of simply generating a word at each time step, the model generates a 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 OpenNMT 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: Variant Attention 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 (), heirarichical 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.

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 work in variational seq2seq auto-encoders (), one-shot learning (), and neural turing machines ().

Table 1: Performance Results. Several languages

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

Image-to-Text As a final case study, we experimented with implementing a complete image-to-text translation system using the OpenNMT library. Qualitatively this is a very different problem than standard machine translation as the source sentence has a different modality. However, it seems like the future of translation may require this style of multi-modal inputs (). In particular, we adapted the im2markup system developed by () to instead use this library. Instead 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

Mainly focus on NMT. Speed, memory, accuracy.
Picture of demo application running

6 Conclusion

References