XNMT: The eXtensible Neural Machine Translation Toolkit

Graham Neubig Carnegie Mellon University, Pittsburgh, USA
Matthias Sperber Karlsruhe Institute of Technology, Karlsruhe, Germany
Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan Carnegie Mellon University, Pittsburgh, USA
Philip Arthur Nara Institute of Science and Technology, Nara, Japan
Pierre Godard LIMSI/CNRS/Université Paris-Saclay, Orsay, France
John Hewitt University of Pennsylvania, Philadelphia, USA
Rachid Riad ENS/CNRS/EHESS/INRIA, Paris, France
Liming Wang University of Illinois, Urbana-Champaign, USA

Abstract

This paper describes XNMT, the eXtensible Neural Machine Translation toolkit. XNMT distinguishes itself from other open-source NMT toolkits by its focus on modular code design, with the purpose of enabling fast iteration in research and replicable, reliable results. In this paper we describe the design of XNMT and its experiment configuration system, and demonstrate its utility on the tasks of machine translation, speech recognition, and multi-tasked machine translation/parsing. XNMT is available open-source at `https://github.com/neulab/xnmt`.

1 Introduction

Due to the effectiveness and relative ease of implementation, there is now a proliferation of toolkits for neural machine translation (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Bahdanau et al., 2015), as many as 51 according to the tally by `nmt-list`.[1] The common requirements for such toolkits are speed, memory efficiency, and translation accuracy, which are essential for the use of such systems in practical translation settings. Many open source toolkits do an excellent job at this to the point where they can be used in production systems (e.g. OpenNMT[2] is used by Systran (Crego et al., 2016)).

This paper describes XNMT, the eXtensible Neural Machine Translation toolkit, a toolkit that optimizes not for efficiency, but instead for ease of use in practical research settings. In other words, instead of only optimizing time for training or inference, XNMT aims to reduce the time it takes for a researcher to turn their idea into a practical experimental setting, test with a large number of parameters, and produce valid and trustable research results. Of course, this necessitates a certain level of training efficiency and accuracy, but XNMT also takes into account a number of considerations, such as those below:

- XNMT places a heavy focus on modular code design, making it easy to swap in and out different parts of the model. Ideally, implementing research prototypes with XNMT involves only few changes to existing code.

[1] `https://github.com/jonsafari/nmt-list`
[2] `http://opennmt.net`

- XNMT is implemented in Python, the de facto standard in the research community.
- XNMT uses DyNet (Neubig et al., 2017) as its deep learning framework. DyNet uses dynamic computation graphs, which makes it possible to write code in a very natural way, and benefit from additional flexibility to implement complex networks with dynamic structure, as are often beneficial in natural language processing. Further benefits include transparent handling of batching operations, or even removing explicit batch handling and relying on autobatching for speed-up instead.
- XNMT of course contains standard NMT models, but also includes functionality for optimization using reinforcement learning (Ranzato et al., 2015) or minimum risk training (Shen et al., 2016), flexible multi-task learning (Dai and Le, 2015), encoders for speech (Chan et al., 2016), and training and testing of retrieval-based models (Huang et al., 2013).

In the remainder of the paper, we provide some concrete examples of the design principles behind XNMT, and a few examples of how it can be used to implement standard models.

2 Model Structure and Specification

2.1 NMT Design Dimensions: Model, Training, and Inference

When training an NMT system there are a number of high-level design decisions that we need to make: what kind of model do we use? how do we test this model? at test time, how do we generate outputs? Each of these decisions has a number of sub-components.

For example, when specifying our model, if we are using a standard attentional model such as that defined by Bahdanau et al. (Bahdanau et al., 2015), we must at least decide:

Input Data Format: Do we use plain text or structured data such as trees?

Embedding: Do we lookup words in a table or encode their characters or other units?

Encoder: Do we use bidirectional LSTMs, convolutional nets, self attention?

Decoder: Do we use standard LSTM-based word-by-word decoders or add tricks such as memory, syntax, or chunks?

Attention: Do we use multi-layer perceptrons, dot-products, or something else?

When specifying the training regimen, there are also choices, including:

Loss Function: Do we use maximum likelihood or a sequence-based training criterion such as REINFORCE or minimum-risk training?

Batching: How many sentences in a mini-batch, and do we sort by length before batching?

Optimizer: What optimization method do we use to update our parameters?

Stopping Criterion: How do we decide when to stop training?

And in inference, there are also options:

Search Strategy: Do we perform greedy search? beam search? random sampling?

Decoding Time Score Adjustment: At decoding time, do we do something like length normalization to give longer hypotheses higher probability?

Within XNMT, effort is made to encapsulate these design decisions in Python classes, making it possible for a researcher who wants to experiment with new alternatives to any of these decisions to implement a new version of the class and compare it with other similar alternatives.

2.2 YAML Model Specification

In order to specify experimental settings, XNMT uses configuration files in YAML[3] format, which provides an easy-to-read, Python-like syntax. An example of such a file, demonstrating

[3] http://yaml.readthedocs.io/en/latest/example.html

```yaml
mini_exp: !Experiment # top of experiment hierarchy
  exp_global: !ExpGlobal # global (default) experiment settings
    model_file: examples/output/{EXP}.mod
    log_file: examples/output/{EXP}.log
    default_layer_dim: 512
    dropout: 0.3
  model: !DefaultTranslator # attentional seq2seq model
    src_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/train.ja.vocab}
    trg_reader: !PlainTextReader
      vocab: !Vocab {vocab_file: examples/data/train.en.vocab}
    src_embedder: !SimpleWordEmbedder {} # {} indicates defaults
    encoder: !BiLSTMSeqTransducer
      layers: 1
    attender: !MlpAttender {}
    trg_embedder: !SimpleWordEmbedder
      emb_dim: 128 # if not set, default_layer_dim is used
    decoder: !MlpSoftmaxDecoder
      layers: 1
      bridge: !CopyBridge {}
  train: !SimpleTrainingRegimen # training strategy
    run_for_epochs: 20
    batcher: !SrcBatcher
      batch_size: 32
    src_file: examples/data/train.ja
    trg_file: examples/data/train.en
    dev_tasks: # what to evaluate at every epoch
      - !LossEvalTask
        src_file: examples/data/dev.ja
        ref_file: examples/data/dev.en
  evaluate: # what to evaluate at the end of training
  - !AccuracyEvalTask
    src_file: examples/data/test.ja
    ref_file: examples/data/test.en
    eval_metrics: bleu
```

Figure 1: Example configuration file

how it is possible to specify choices along the various design dimensions in §2.1 is shown in Figure 1.[4] As shown in the example, XNMT configuration files specify a hierarchy of objects, with the top level always being an `Experiment` including specification of the model, training, and evaluation, along with a few global parameters shared across the various steps.

One thing that is immediately noticeable from the file is the `!` syntax, which allows to directly specify Python class objects inside the YAML file. For any item in the YAML hierarchy that is specified in this way, all of its children in the hierarchy are expected to be the arguments to its constructor (the Python method). So for example, if a user wanted to test create a method for convolutional character-based encoding of words (Zhang et al., 2015) and see its result on machine translation, they would have to define a new class `ConvolutionalWordEmbedder(filter_width, embedding_size=512)`, implement it appropriately, then in the YAML file replace the `src_embedder:` line with
optionally omitting `embedding_size:` if the defaults are acceptable.

[4]For many of these parameters XNMT has reasonable defaults, so the standard configuration file is generally not this verbose.

```
src_embedder: !ConvolutionalWordEmbedder
  filter_width: 3
  embedding_size: 512
```

As may be evident from the example, this greatly helps extensibility for two reasons: (1) there is no passing along of command line arguments or parsing of complex argument types necessary. Instead, objects are simply configured via their Python interface as given in the code, and newly added features can immediately be controlled from the configuration file without extra argument handling. (2) Changing behavior is as simple as adding a new Python class, implementing the required interface, and requesting the newly implemented class in the configuration file instead of the original one.

2.3 Experimental Setup and Support

As Figure 1 demonstrates, XNMT supports the basic functionality for experiments described in §2.1. In the example, the model specifies the input data structure to be plain text (`PlainTextReader`), word embedding method to be a standard lookup-table based embedding (`SimpleWordEmbedder`), encoder to be a bidirectional LSTM (`BiLSTMSeqTransducer`), attender to be a multi-layer perceptron based attention method (`MlpAttender`), and the decoder to use a LSTM with a MLP-based softmax (`MlpSoftmaxDecoder`). Similarly, in the `training:` and `evaluate:` subsections, the training and evaluation parameters are set as well.

XNMT also provides a number of conveniences to support efficient experimentation:

Named experiments and overwriting: Experiments are given a name such as `mini_exp`. `{EXP}` strings in the configuration file are automatically overwritten by this experiment name, distinguishing between log or model files from different experiments.

Multiple experiments and sharing of parameters: Multiple experiments can be specified in a single YAML file by defining multiple top-level elements of the YAML file. These multiple experiments can share settings through YAML anchors, where one experiment can inherit the settings from another, only overwriting the relevant settings that needs to be changed.

Saving configurations: For reproducibility, XNMT dumps the whole experiment specification when saving a model. Thus, experiments can be re-run by simply opening the configuration file associated with any model.

Re-starting training: A common requirement is loading a previously trained model, be it for fine-tuning on different data, tuning decoding parameters, or testing on different data. XNMT allows this by re-loading the dumped configuration file, overwriting a subset of the settings such as file paths, decoding parameters, or training parameters, and re-running the experiment.

Random parameter search: Often we would like to tune parameter values by trying several different configurations. Currently XNMT makes it possible to do so by defining a set of parameters to evaluate and then searching over them using random search. In the future, we may support other strategies such as Bayesian optimization or enumeration.

3 Advanced Features

3.1 Advanced Modeling Techniques

XNMT provides a wide library of standard modeling tools of use in performing NMT such as speech-oriented encoders (Chan et al., 2016; Harwath et al., 2016) that can be used in speech recognition, preliminary support for self-attentional "Transformer" models (Vaswani et al., 2017). It also has the ability to perform experiments in retrieval (Huang et al., 2013) instead of sequence generation.

```
tied_exp: !Experiment
  ...
  model: !DefaultTranslator
    ..
    trg_embedder: !DenseWordEmbedder
      emb_dim: 128
    decoder: !MlpSoftmaxDecoder
      layers: 1
      bridge: !CopyBridge {}
      vocab_projector: !Ref { path: model.trg_embedder }
```

Figure 2: Illustration of referencing mechanism

3.2 Parameter Sharing and Multi-task Learning

Modern deep learning architectures often include parameter sharing between certain components. For example, tying the output projection matrices and embeddings has been proposed by Press and Wolf (2016). While it would be possible to develop a specialized component to achieve this, XNMT features a referencing mechanism that allows simply tying the already existing components (Figure 2). References are created by specifying the path of the object to which they point, and result in the exact same object instance being used in both places. The only requirement is for the object's interface to be compatible with both usages, which is usually easily achieved using Python's duck typing coding paradigm.

This component sharing is also very useful in multi-task training paradigms, where two tasks are trained simultaneously and share some or all of their component parts. This multi-task training can be achieved by replacing the `SimpleTrainingRegimen` with other regimens specified for multi-task learning, and defining two or more training tasks that use different input data, models, or training parameters.

3.3 Training and Inference Methods

XNMT provides several advanced methods for training and inference. With regards to training, XNMT notably makes it easy to implement other training criteria such as REINFORCE or minimum risk training by defining a separate class implementing the training strategy. REINFORCE has been implemented, and more training criteria may be added in the near future. For inference, it is also possible to specify several search strategies (e.g. beam search), along with several length normalization strategies that helps reduce the penalty on long sentences.

4 Case Studies

In this section, we describe three case studies of using XNMT to perform various experiments: a standard machine translation experiment (§4.1), a speech recognition experiment (§4.2), and a multi-task learning experiment where we train a parser along with an MT model (§4.3).

4.1 Machine Translation

We trained a machine translation model on the WMT English-German benchmark, using the preprocessed data by Stanford.[5] Our model was a basic 1-layer model with bidirectional LSTM encoder and 256 units per direction, LSTM decoder output projections and MLP attention mechanism all with 512 hidden units. We applied joint BPE of size 32k (Sennrich et al., 2016). We also applied input feeding, as well as variational dropout of rate 0.3 to encoder and decoder LSTMs. Decoding was performed with a beam of size 1. Overall, results were similar, with our model achieving a BLEU of 18.26 and Luong et al. (2015) achieving a BLEU of 18.1. Note that the model by Luong et al. (2015) is simpler because it does not use BPE and only a unidirectional encoder.

[5] `https://nlp.stanford.edu/projects/nmt/`

Model	WSJ dev93	WSJ eval92	TEDLIUM dev	TEDLIUM test
XNMT	16.65	13.50	15.83	16.16
Zhang et al. (2017)	—	14.76	—	
Rousseau et al. (2014)	—	—	15.7	17.8

Table 1: Speech recognition results (WER in %) compared to a similar pyramidal LSTM model (Zhang et al., 2017) and a highly engineered hybrid HMM system (Rousseau et al., 2014).

4.2 Speech Recognition

We performed experiments in a speech recognition task with a simple listen-attend-spell model (Chan et al., 2016). This model features a 4-layer pyramidal LSTM encoder, subsampling the input sequence by factor 2 at every layer except the first, resulting in an overall subsampling factor of 8. The layer size is set to 512, the target embedding size is 64, and the attention uses an MLP of size 128. Input to the model are Mel-filterbank features with 40 coefficients. For regularization, we apply variational dropout of rate 0.3 in all LSTMs, and word dropout of rate 0.1 on the target side (Gal and Ghahramani, 2016). For training, we use Adam (Kingma and Ba, 2014) with initial learning rate of 0.0003, which is decayed by factor 0.5 if no improved in WER is observed. To further facilitate training, label smoothing (Szegedy et al., 2016) is applied. For the search, we use beam size 20 and length normalization with the exponent set to 1.5. We test this model on both the Wall Street Journal (WSJ; Paul and Baker (1992)) corpus which contains read speech, and the TEDLIUM corpus (Rousseau et al., 2014) which contains recorded TED talks. Numbers are shown in Table 4.2. Comparison to results from the literature shows that our results are competitive.

4.3 Multi-task MT + Parsing

We performed multi-task training of a sequence-to-sequence model for parsing and machine translation. The main task is the parsing task, and we followed the general setup in (Vinyals et al., 2015), but we only used the standard WSJ training data. It is jointly trained with an English-German translation system. Compared to a single sequence-to-sequence model for parsing with the same hyperparameters as the multi-task model, a model trained only on WSJ achieved a test F-score of 81%, while the multi-task trained model achieved an F-score of 83%. This experiment was done with very few modifications to existing XNMT multi-task architecture, demonstrating that it is relatively easy to apply multi-tasking to new tasks.

5 Conclusion

This paper has introduced XNMT, an NMT toolkit with extensibility in mind, and has described the various design decisions that went into making this goal possible.

Acknowledgments

Part of the development of XNMT was performed at the Jelinek Summer Workshop in Speech and Language Technology (JSALT) “Speaking Rosetta Stone” project (Scharenborg et al., 2018), and we are grateful to the JSALT organizers for the financial/logistical support, and also participants of the workshop for their feedback on XNMT as a tool.

Parts of this work were sponsored by Defense Advanced Research Projects Agency Information Innovation Office (I2O). Program: Low Resource Languages for Emergent Incidents (LORELEI). Issued by DARPA/I2O under Contract No. HR0011-15-C-0114. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *Proc. of ICLR*.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE.

Crego, J., Kim, J., Klein, G., Rebollo, A., Yang, K., Senellart, J., Akhanov, E., Brunelle, P., Coquard, A., Deng, Y., et al. (2016). Systran's pure neural machine translation systems. *arXiv preprint arXiv:1610.05540*.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3079–3087.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

Harwath, D., Torralba, A., and Glass, J. (2016). Unsupervised learning of spoken language with visual context. In *Advances in Neural Information Processing Systems*, pages 1858–1866.

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent Continuous Translation Models. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1700–1709, Seattle, Washington, USA.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastasopoulos, A., Ballesteros, M., Chiang, D., Clothiaux, D., Cohn, T., Duh, K., Faruqui, M., Gan, C., Garrette, D., Ji, Y., Kong, L., Kuncoro, A., Kumar, G., Malaviya, C., Michel, P., Oda, Y., Richardson, M., Saphra, N., Swayamdipta, S., and Yin, P. (2017). DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*.

Paul, D. B. and Baker, J. M. (1992). The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics.

Press, O. and Wolf, L. (2016). Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2015). Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Rousseau, A., Deléglise, P., and Estève, Y. (2014). Enhancing the ted-lium corpus with selected data for language modeling and more ted talks. In *LREC*, pages 3935–3939.

Scharenborg, O., Besacier, L., Black, A., Hasegawa-Johnson, M., Metze, F., Neubig, G., Stuker, S., Godard, P., Muller, M., Ondel, L., Palaskar, S., Arthur, P., Ciannella, F., Du, M., Larsen, E., Merkx, D., Riad, R., Wang, L., and Dupoux, E. (2018). Linguistic unit discovery from multi-modal inputs in unwritten languages: Summary of the "speaking rosetta" JSALT 2017 workshop. In *2018 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2018)*, Calgary, Canada.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *54th Annual Meeting of the Association for Computational Linguistics*.

Shen, S., Cheng, Y., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692, Berlin, Germany. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, Montréal, Canada.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. E. (2015). Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2773–2781.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Zhang, Y., Chan, W., and Jaitly, N. (2017). Very deep convolutional networks for end-to-end speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4845–4849. IEEE.