THUMT: An Open-Source Toolkit for Neural Machine Translation

The Tsinghua Natural Language Processing Group

January 1, 2018

1 Introduction

Machine translation, which investigates the use of computer to translate human languages automatically, is an important task in natural language processing and artificial intelligence. With the availability of bilingual, machine-readable texts, data-driven approaches to machine translation have gained wide popularity since 1990s. Recent several years have witnessed the rapid development of end-to- end neural machine translation (NMT) (Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017). Capable of learning representations from data, NMT has quickly replaced conventional statistical machine translation (SMT) (Brown et al., 1993; Koehn et al., 2003; Chiang, 2005) to become the new *de facto* method in practical MT systems (Wu et al., 2016).

THUMT is an open-source toolkit for neural machine translation developed by the Tsinghua Natural Language Processing Group. It currently has two main implementations:

1. THUMT-TesnorFlow[1]: a new implementation developed with TensorFlow.[2] It implements the sequence-to-sequence (Seq2Seq) (Sutskever et al., 2014), the standard attention-based model (RNNsearch) (Bahdanau et al., 2015), and the Transformer model (Transformer) (Vaswani et al., 2017). The training criterion is maximum likelihood estimation (MLE).

2. THUMT-Theano[3]: the original project developed with Theano[4], which is no longer updated because MLA put an end to Theano. It implements the RNNsearch model, minimum risk training (MRT) (Shen et al., 2016) for optimizing model parameters with respect to evaluation metrics, semi-supervised training (SST) (Cheng et al., 2016) for exploiting monolingual corpora to learn bidirectional translation models, and layer-wise relevance propagation (LRP) (Ding et al., 2017) for visualizing and analyzing RNNsearch.

[1] https://github.com/thumt/THUMT
[2] http://tensorflow.org
[3] https://github.com/thumt/THUMT/tree/theano
[4] https://github.com/Theano/Theano

		THUMT-Theano	THUMT-TensorFlow
Model	Seq2Seq	×	√
	RNNsearch	√	√
	Transformer	×	√
Criterion	MLE	√	√
	MRT	√	×
	SST	√	×
LRP	Seq2Seq	×	×
	RNNsearch	√	×
	Transformer	×	×

Table 1: Two implementations of THUMT.

Table 1 summarizes the the features of the two implementations.

We recommend using THUMT-TensorFlow, which delivers better translation performance than THUMT-Theano. We will keep adding new features to THUMT-TensorFlow. This document is the user manual for THUMT-TensorFlow. For simplicity, we will refer to THUMT-TesnorFlow as THUMT in the following sections.

2 Installation

2.1 System Requirements

THUMT supports Linux i686 and Mac OSX. The following third-party software and toolkits are required to install and use THUMT:

1. `Python` v2.7.0 or higher;
2. `TensorFlow` v1.4.0 or higher.

2.2 Installing THUMT

The source code of THUMT is available at https://github.com/thumt/THUMT. Here is a brief guide on how to install THUMT.

2.2.1 Step1: Downloading the Package

Download the package using the following command:

```
1 $ git clone https://github.com/thumt/THUMT.git
```

Entering the `THUMT` folder, you may find two folders (`thumt`, `docs`) and three files (`LICENSE`, `README.md`, `UserManual.pdf`):

1. `thumt`: the source code;

2. `docs`: LaTex files of the user manual;
3. `LICENSE`: license statement;
4. `README.md`: the readme formatted with Markdown;
5. `UserManual.pdf`: this document.

2.2.2 Step 2: Modifying Environment Variables

We highly recommend running THUMT on GPU servers. Suppose THUMT runs on NVIDIA GPUs with the CUDA toolkit v8.0 installed. Users need to set environment variables to enable the GPU support:

```
$ export PATH=/usr/local/cuda/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
$ export PYTHONPATH=/PATH/TO/THUMT:$PYTHONPATH
```

To set these environment variable permanently for all future bash sessions, users can simply add the above three lines to the `.bashrc` file in your `$HOME` directory, where "`/PATH/TO/THUMT`" is the path to the `THUMT` folder.

3 User Guide

3.1 Data Preparation

3.1.1 Obtaining the Datasets

Running THUMT involves three types of datasets:

1. *Training set*: a set of parallel sentences used for training NMT models.
2. *Validation set*: a set of source sentences paired with single or multiple target translations used for model selection and hyper-parameter optimization.
3. *Test set*: a set of source sentences paired with single or multiple target translations used for evaluating translation performance on unseen texts.

In this user manual, we take the preprocessed German-English dataset of the WMT 2017 news translation shared task as an example.[5] Download and unpack the two files `corpus.tc.de.gz` and `corpus.tc.en.gz`:

```
$ gunzip corpus.tc.de.gz corpus.tc.en.gz
```

The resulting files `corpus.tc.de` and `corpus.tc.en` serve as the training set,

[5] http://data.statmt.org/wmt17/translation-task/preprocessed/de-en/

which contains 5,852,458 pairs of sentences. Note that the German and English sentences are preprocessed using tokenization and true casing.

Unpack the file `dev.tgz` using the following command:

```
$ tar xvfz dev.tgz
```

In the `dev` folder, `newstest2014.tc.de` and `newstest2014.tc.en` serve as the validation set, which contains 3,003 pairs of sentences. The test set we use is `newstest2015.tc.de` and `newstest2015.tc.en`, which consists of 2,169 pairs of sentences. Note that both the validation and test sets use single references since there is only one gold-standard English translation for each German sentence.

3.1.2 Running BPE

For efficiency reasons, only a fraction of the full vocabulary can be used in neural machine translation systems. The most widely used approach for addressing the open vocabulary problem is to use the Byte Pair Encoding (BPE) (Sennrich et al., 2016). We recommend using BPE for THUMT.

First, download the source code of BPE using the following command:

```
$ git clone https://github.com/rsennrich/subword-nmt.git
```

To encode the training corpora using BPE, you need to generate BPE operations first. The following command will create a file named `bpe32k`, which contains 32k BPE operations. It also outputs two dictionaries named `vocab.de` and `vocab.en`.

```
$ python subword-nmt/learn_joint_bpe_and_vocab.py --input
corpus.tc.de corpus.tc.en -s 32000 -o bpe32k --write-vocabulary
vocab.de vocab.en
```

Then, the `apply_bpe.py` script runs to encode the training set using the generated BPE operations and dictionaries.

```
$  python subword-nmt/apply_bpe.py --vocabulary vocab.de
--vocabulary-threshold 50 -c bpe32k < corpus.tc.de >
corpus.tc.32k.de
$ python subword-nmt/apply_bpe.py --vocabulary vocab.en
--vocabulary-threshold 50 -c bpe32k < corpus.tc.en >
corpus.tc.32k.en
```

The validation set also needs to be processed using the `apply_bpe.py` script.

```
$ python subword-nmt/apply_bpe.py --vocabulary vocab.de
--vocabulary-threshold 50 -c bpe32k < newstest2014.tc.de >
newstest2014.tc.32k.de
$ python subword-nmt/apply_bpe.py --vocabulary vocab.en
--vocabulary-threshold 50 -c bpe32k < newstest2014.tc.en >
newstest2014.tc.32k.en
```

Finally, the source side of the test side is processed as well.

```
$ python subword-nmt/apply_bpe.py --vocabulary vocab.de
--vocabulary-threshold 50 -c bpe32k < newstest2015.tc.de >
newstest2015.tc.32k.de
```

3.1.3 Shuffling Training Set

The next step is to shuffle the training set, which proves to be helpful for improving the translation quality. Simply run the following command:

```
$ python THUMT/thumt/scripts/shuffle_corpus.py --corpus
corpus.tc.32k.de corpus.tc.32k.en --suffix shuf
```

The resulting files `corpus.tc.32k.de.shuf` and `corpus.tc.32k.en.shuf` rearrange the sentence pairs randomly.

3.1.4 Generating Vocabularies

We need to generate vocabulary from the shuffled training set. This can be done by running the `build_vocab.py` script:

```
$python THUMT/thumt/scripts/build_vocab.py corpus.tc.32k.de.shuf
vocab.32k.de
$python THUMT/thumt/scripts/build_vocab.py corpus.tc.32k.en.shuf
vocab.32k.en
```

The resulting files `vocab.32k.de.txt` and `vocab.32k.en.txt` are final source and target vocabularies used for model training.

3.2 Training

3.2.1 Transformer

We recommend using the Transformer model that delivers the best translation performance among all the three models supported by THUMT.

The command for training a Transformer model is given by

```
$python THUMT/thumt/bin/trainer.py --input corpus.tc.32k.de.shuf
corpus.tc.32k.en.shuf --vocabulary vocab.32k.de.txt
vocab.32k.en.txt --model transformer --validation
newstest2014.tc.32k.de --references newstest2014.tc.32k.en
--parameters=batch_size=6250,device_list=[0],train_steps=200000
```

Note that we set the `batch_size` to 6,250 words instead of 6,250 sentences. By default, the batch size for the Transformer model is defined in terms of word number rather than sentence number in THUMT.

"`device_list=[0]`" suggests that `gpu0` is used to train the model. THUMT supports to train NMT models on multiple GPUs. If both `gpu0` and `gpu1` are available, simply set "`device_list=[0,1]`". You may use the `nvidia-smi` command to find unused GPUs.

By setting "`train_steps=200000`", the training process will terminate at iteration 200,000. During the training, the `trainer.py` script creates a `train` folder to store intermediate models called *checkpoints*, which will be evaluated on the validation set periodically. Only a small number of checkpoints that achieves highest BLEU scores on the validation set will be saved in the `train/eval` folder. This folder will be used in decoding.

3.2.2 RNNsearch

The command for training an RNNsearch model is given by

```
$python THUMT/thumt/bin/trainer.py --input corpus.tc.32k.de.shuf
corpus.tc.32k.en.shuf --vocabulary vocab.32k.de.txt
vocab.32k.en.txt --model rnnsearch --validation
newstest2014.tc.32k.de --references newstest2014.tc.32k.en
--parameters=batch_size=128,device_list=[0],train_steps=200000
```

Note that we set the `batch_size` to 128 sentences instead of 128 words. By default, the batch size for the RNNsearch model is defined in terms of sentence number rather than word number in THUMT. The trained models are also saved in the `train/eval` folder.

3.2.3 Seq2Seq

The command for training a Seq2Seq model is given by

```
$python THUMT/thumt/bin/trainer.py --input corpus.tc.32k.de.shuf
corpus.tc.32k.en.shuf --vocabulary vocab.32k.de.txt
vocab.32k.en.txt --model seq2seq --validation
newstest2014.tc.32k.de --references newstest2014.tc.32k.en
--parameters=batch_size=128,device_list=[0],train_steps=200000
```

Note that we set the `batch_size` to 128 sentences instead of 128 words. By default, the batch size for the Seq2Seq model is also defined in terms of sentence number rather than word number in THUMT. The trained models are saved in the `train/eval` folder as well.

3.3 Decoding

The command for translating the test set using the trained Transformer model is given by

```
$ python THUMT/thumt/bin/translator.py --models transformer
--input newstest2015.tc.32k.de --output newstest2015.trans
--vocabulary vocab.32k.de vocab.32k.en --checkpoints train/eval
--parameters=device_list[0]
```

The commands for using RNNsearch and Seq2Seq models are similar except for the `--models` argument. The translation file output by the `translator.py` is `newstest2015.trans`, which needs to be restored to the normal tokenization using the following command:

```
sed -r 's/(@@ )|(@@ ?$)//g' < newstest2015.trans >
newstest2015.trans.norm
```

Finally, BLEU scores (Papineni et al., 2002) can be calculated using the `multi-bleu.perl`[6]:

```
$ multi-bleu.perl -lc newstest2015.tc.en
< newstest2015.trans.norm > evalResult
```

The resulting `evalResult` stores the calculated BLEU scores.

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

4 FAQ

4.1 Does THUMT support model averaging?

Yes. THUMT uses the `checkpoint_averaging.py` script to average checkpoints generated during training to generate a better model. The script is located in the `THUMT/thumt/scripts` folder. Simply run the following command:

```
$python THUMT/thumt/scripts/checkpoint_averaging.py --path train
--checkpoints 5 --output averaged
```

By default, the `train` folder saves the last 20 checkpoints. The above command averages 5 of them and saves the averaged model in the `averaged` folder. You may change the number of checkpoints by modifying the `--checkpoints` argument.

4.2 Does THUMT support model ensemble?

Yes. Suppose THUMT runs for four times on the same training data. Due to random initialization, the four trained models are different. The following command can be used to ensemble the four models located in four separated folders:

```
$ python THUMT/thumt/bin/translator.py --models transformer
--input newstest2015.tc.32k.de --output newstest2015.trans
--vocabulary vocab.32k.de vocab.32k.en
--checkpoints run1/train run2/train run3/train run4/train
--parameters=device_list[0]
```

4.3 How to save disk space during training?

During training, latest checkpoints are stored in the `train` folder. The number of checkpoints is specified by the `keep_checkpoint_max` parameter in the `trainer.py` script. Its default value is 20.

Checkpoints with top BLEU scores on the validation set are stored in the `train/eval` folder. The number is specified by the `keep_top_checkpoint_max` parameter in the `trainer.py` script. Its default value is 5.

The size of a checkpoint depends is usually about 1GB. As a result, by default, the `train` folder that stores intermediate files often takes up 30GB of disk space. To save disk space, simply set the two parameters to smaller values. For example, we can set `keep_checkpoint_max` to 5 to keep only 5 checkpoints in the `train` folder and `keep_top_checkpoint_max` to 1 to keep only 1 checkpoint in the `train/eval` folder. Note that `keep_checkpoint_max` also determines the maximum number of checkpoints that can be used for model averaging.

An example command is shown below:

```
$python THUMT/thumt/bin/trainer.py --input corpus.tc.32k.de.shuf
corpus.tc.32k.en.shuf --vocabulary vocab.32k.de.txt
vocab.32k.en.txt --model transformer --validation
newstest2014.tc.32k.de --references newstest2014.tc.32k.en
--parameters=batch_size=6250,device_list=[0],train_steps=200000,
keep_checkpoint_max=5,keep_top_checkpoint_max=1
```

A Parameters of The trainer.py Script

The parameters of the `trainer.py` scripts is specified by a string containing comma-separated `name=value` pairs. Here, we list important parameters in detail.

A.1 General Parameters

1. `num_threads`: the number of threads used in data processing. The default value is 6.

2. `buffer_size`: the buffer size used in data processing. The default value is 10,000.

3. `batch_size`: the batch size used in the training stage. The default value is 128.

4. `constant_batch_size`: a boolean value that specifies whether the number of sentences is treated as `batch_size`. The default value is `true`. If it is set to `true`, `batch_size` is defined as the number of sentences. This usually happens for training Seq2Seq and RNNsearch models. If it is set to `false`, `batch_size` is defined as the number of tokens, which is more appropriate for training the Transformer model.

5. `max_length`: the maximum length of a sentence in the training set. The default value is 256.

6. `train_steps`: the total number of steps in the training stage. The default value is 100,000.

7. `update_cycle`: the number of iterations for updating model parameters. The default value is 1. If you have only 1 GPU and want to obtain the same translation performance with using 4 GPUs, simply set this parameter to 4. Note that the training time will also be prolonged.

8. `save_checkpoint_steps`: the number of steps for saving a checkpoint periodically. The default value is 1,000.

9. `initializer`: choose how to initialize model parameters. Possible values are `uniform`, `normal`, `uniform_unit_scaling`, and `normal_unit_scaling`. The default value is `uniform`.

10. `initializer_gain`: set the parameter of the initializer. The default value is 0.08.

11. `learning_rate`: set the learning rate. The default value is 1.0.

12. `learning_rate_decay`: set learning rate decay function. Possible values are `noam`, `piecewise_constant`, and `none`. The default value is `noam`.

13. `learning_rate_boundaries`: learning rate boundaries. The default value is [0].

14. `learning_rate_values`: learning rate values. The default value is [0.0].

15. `keep_checkpoint_max`: the maximum number of checkpoints to keep during training. The default value is 20.

16. `keep_top_checkpoint_max`: the maximum number of top performed checkpoints to keep during training. The default value is 5.

17. `eval_steps`: the number of steps for evaluating the intermediate model periodically on the validation set. The default value is 2,000.

18. `eval_batch_size`: the batch size for evaluating the intermediate model periodically on the validation set. The default value is 32.

19. `beam_size`: beam size for beam search in decoding. The default value is 4.

20. `decode_alpha`: the length penalty term in the beam search (Wu et al., 2016). The default value is 0.6.

21. `decode_length`: the maximum length ratio of a translation. The default value is 50.

22. `device_list`: the list of GPUs to be used in training. Use the `nvidia-smi` command to find unused GPUs. If the unused GPUs are `gpu0` and `gpu1`, set this parameter as `device_list=[0,1]`.

A.2 Parameters for Seq2Seq

1. `rnn_cell`: the recurrent unit. Possible values are `LSTMCell` and `GRUCell`. The default value is `LSTMCell`.

2. `embedding_size`: word embedding size for source and target languages. The default value is 1,000.

3. `hidden_size`: the size of hidden layers. The default value is 1,000.

4. `num_hidden_layers`: the number of hidden layers. the default value is 4.
5. `dropout`: dropout rate. The default value is 0.2.
6. `label_smoothing`: the value of label smoothing. The default value is 0.1.
7. `reverse_source`: a boolean value that specifies whether the input sentence is reversed (Sutskever et al., 2014). The default value is `true`.
8. `use_residual`: a boolean value that specifies whether residual connections are used for multi-layered RNNs. The default value is `true`.

A.3 Parameters for RNNsearch

1. `rnn_cell`: the recurrent unit. Currently, only GRU is supported. The default value is `LegacyGRUCell`.
2. `embedding_size`: word embedding size for source and target languages. The default value is 620.
3. `hidden_size`: the size of hidden layers. The default value is 1,000.
4. `maxnum`: the hidden units of maxout layer. The default value is 2.
5. `dropout`: dropout rate. The default value is 0.2.
6. `label_smoothing`: the value of label smoothing. The default value is 0.1.

A.4 Parameters for Transformer

1. `hidden_size`: the embedding size and hidden size of the network. The default value is 512.
2. `filter_size`: the hidden size of the feed-forward layer. The default value is 2,048.
3. `num_encoder_layers`: the number of encoder layers. The default value is 6.
4. `num_decoder_layers`: the number of decoder layers. The default value is 6.
5. `num_heads`: the number of attention heads used in the multi-head attention mechanism. The default value is 8.
6. `shared_embedding_and_softmax_weights`: a boolean value that specifies whether to share the embedding and softmax weights. The default value is `false`.
7. `shared_source_target_embedding`: a boolean value that specifies whether to share the source and target embeddings. The default value is `false`.

8. `residual_dropout`: the dropout rate used in residual connection. The default value is 0.1.

9. `attention_dropout`: the dropout rate used in attention mechanism. The default value is 0.0.

10. `relu_dropout`: the dropout rate used in feed forward layer. The default value is 0.0.

11. `label_smoothing`: the value of label smoothing. The default value is 0.1.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*.

Cheng, Y., Xu, W., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Semi-supervised learning for neural machine translation. In *Proceedings of ACL*.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL*.

Ding, Y., Liu, Y., Luan, H., and Sun, M. (2017). Visualizing and understanding neural machine translation. In *Proceedings of ACL*.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of NAACL*.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of ACL*.

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 ACL*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of NIPS*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of NIPS*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144v2.