THUMT: An Open-Source Toolkit for Neural Machine Translation

The Tsinghua Natural Language Processing Group

December, 2017

1 Introduction

Machine translation, which investigates the use of computer to translate human languages automatically, is an important task in natural language processing an 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). 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).

On top of TensorFlow, THUMT is an open-source toolkit for neural machine translation developed by the Tsinghua Natural Language Processing Group. THUMT has the following features:

1. *Attention-based translation model.* THUMT implements the standard attention-based encoder-decoder framework for NMT(Bahdanau et al., 2015) as well as the latest Transformer architecture (Vaswani et al., 2017).

2. *Minimum risk training.* Besides standard maximum likelihood estimation (MLE), THUMT also supports minimum risk training (MRT) (Shen et al., 2016) that aims to find a set of model parameters that minimize the expected loss calculated using evaluation metrics such as BLEU (Papineni et al., 2002) on the training data.

2 Installation

2.1 System Requirements

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

1. Python version 2.7.0 or higher.

2. JRE 1.6 or higher (optional, only used for visualization).

2.2 Installing Prerequisites

We recommend using pip to install the prerequisites of THUMT. The installation starts with python-pip:

```
apt-get install python-pip
```

Then, run the following two commands to install argparse and TensorFlow (version $\geq$ 1.4.0):

```
pip install argparse
pip install tensorflow-gpu
```

2.3 Installing THUMT

The source code of THUMT is available both at the toolkit website (stable release) and GitHub (latest version).

Here is a brief guide on how to install THUMT.

2.3.1 Step1: Unpacking

Unpack the package using the following command:

```
tar xvfz THUMT.tar.gz
```

Entering the `THUMT` folder, you may find two folders (`thumt`, `data`) and three files (`LICENSE`, `UserManual.en.pdf` and `UserManual.zh.pdf`):

1. `thumt`: the source code.
2. `data`: toy training, validation, and test datasets.
3. `docs`: source of the documents.
4. `LICENSE`: license statement.
5. `UserManual.en.pdf`: this document.
6. `UserManual.zh.pdf`: the Chinese version of this document.

2.3.2 Step 2: Modifying Environment Variables

We highly recommend running THUMT on GPU servers. Suppose THUMT runs on NVIDIA GPUs with the CUDA toolkit version 8.0 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 two lines to the `.bashrc` file in your `$HOME` directory, where `/PATH/TO/THUMT` is the path of `THUMT`.

3 User Guide

3.1 Data Preparation

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.

3.1.1 Training Set

A training set is used for training NMT models. It often consists of two files: one file for source sentences and the other for corresponding target sentences. In the `data` folder, there is an example source file `train.src` of a toy training set that contains seven sentences[1]:

```
wo hen xihuan yinyue .
wo bu xihuan huahua .
ni xihuan yinyue me ?
shide , wo ye xihuan yiyue .
ta ye xihuan yinyue .
ta yidian dou bu xihuan yiyue .
ta hen xinhuan huahua .
```

The corresponding target file `train.trg` is shown below:

```
i like music very much .
i do not like painting .
do you like music ?
yes , i like music too .
he also likes music .
she does not like music at all .
she likes painting very much .
```

Note that each line in the source and target files only contains one tokenized sentence. The source and target sentences with the same line number are translationally equivalent.

[1] The Chinese text is romanized for readability.

Our toy training set only contains seven sentence pairs. In practice, NMT often requires millions of sentence pairs to achieve reasonable translation performance.

3.1.2 Validation Set

A validation set is used for model selection and hyper-parameter optimization. During training, THUMT evaluates intermediate models on the validation set periodically. The model that obtains the highest BLEU score on the validation set is chosen as the final learned model. A validation set consists of one source file and one or more target files. There is an example source file `valid.src` of a toy validation set in the `data` folder:

```
wo hen xihuan huahua .
wo bu xihuan yinyue .
```

In our toy validation set, there is only one target file `valid.trg` that contains the reference translations of `valid.src`:

```
i like painting very much .
i do not like music .
```

Due to the diversity of natural languages, one source sentence often corresponds to multiple reference translations. THUMT also supports multiple references.

3.1.3 Test set

A test set is used for evaluating the learned NMT model on unseen source text. Like a validation set, a test set also contains one source file and one or more target files. It shares the same naming scheme for multiple references with the validation set.

In the `data` folder, there is one source file `test.src`:

```
ta xihuan huahua me ?
ta yidian dou bu xihuan huahua .
```

The corresponding target file `test.trg` is shown below:

```
does she like painting ?
he does not like painting at all .
```

In our toy data, validation and test sets contain only two sentences. In practice, thousands of sentences are needed for both validation and test sets.

3.2 Training

3.3 Data preprocessing

The most common approach to achieve open vocabularies is to use the Byte Pair Encoding (BPE). The codes of BPE can be found at Subword-NMT.

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.src` and `vocab.trg`.

```
python subword-nmt/learn_joint_bpe_and_vocab.py
  --input train.src train.trg -s 32000 -o bpe32k
  --write-vocabulary vocab.src vocab.trg
```

You also need to encode the training corpora, validation set and test set using the generated BPE operations and dictionaries.

```
python subword-nmt/apply_bpe.py
    --vocabulary vocab.src
    --vocabulary-threshold 50
    -c bpe32k < train.src > train.32k.src
  python subword-nmt/apply_bpe.py
    --vocabulary vocab.trg
    --vocabulary-threshold 50
    -c bpe32k < train.trg > train.32k.trg
  python subword-nmt/apply_bpe.py
    --vocabulary vocab.src
    --vocabulary-threshold 50
    -c bpe32k < valid.src > valid.32k.src
  python subword-nmt/apply_bpe.py
    --vocabulary vocab.trg --vocabulary-threshold 50
    -c bpe32k < valid.trg > valid.32k.trg
  python subword-nmt/apply_bpe.py
    --vocabulary vocab.src
    --vocabulary-threshold 50
    -c bpe32k < test.src > test.32k.src
```

The original segmentation can be restored with the following command:

```
sed -r 's/(@@ )|(@@ ?$)//g' < input > output
```

3.4 Shuffling Corpora

It is helpful to shuffle training corpora first and we provide `shuffle_corpus.py` script to achieve this functionality, which has the following options

```
usage: shuffle_corpus.py [-h] --corpus CORPUS [CORPUS ...]
                         [--suffix SUFFIX]
                         [--seed SEED]

Shuffle corpus

optional arguments:
  -h, --help            show this help message and exit
```

```
  --corpus CORPUS [CORPUS ...]
                        input corpora
  --suffix SUFFIX       Suffix of output files
  --seed SEED           Random seed
```

The meaning of arguments is shown below:

1. `corpus`: the input corpus.
2. `--suffix`the suffix of output filename.
3. `--seed`: the random seed.
4. `--help`show the help message.

3.4.1 Generating Vocabularies

We have to build vocabularies before training an NMT model. The `build_vocab.py` in the `scripts` folder can be used to create vocabularies. It has the following options:

```
usage: build_vocab.py [-h] [--limit LIMIT] [--control CONTROL]
                      corpus output

Create vocabulary

positional arguments:
  corpus             input corpus
  output             Output vocabulary name

optional arguments:
  -h, --help         show this help message and exit
  --limit LIMIT      Vocabulary size
  --control CONTROL  Add control symbols to vocabulary.
                     Control symbols are separated by comma.
```

The meaning of arguments is shown below:

1. `corpus`: path to source or target corpus.
2. `output`: the name of output vocabulary.
3. `--limit`: control the size of vocaublary.
4. `--control`: a comma separated string represents the control symbols. For example, the string "`<pad>,<eos>,<unk>`" will add three control symbols to the beginning of the vocabulary.
5. `--help`: show the help message.

3.4.2 The Python Script for Training

THUMT implements the traditional sequence to sequence model SEQ2SEQ (Sutskever et al., 2014), standard attention-based encoder-decoder framework for neural machine translation RNNSEARCH (Bahdanau et al., 2015) as well as the attention-based Transformer architecture (Vaswani et al., 2017).

The `trainer.py` script in the `bin` folder is used for training NMT models. Its arguments are listed as follows:

```
usage: trainer.py [<args>] [-h | --help]

Training neural machine translation models

optional arguments:
  -h, --help            show this help message and exit
  --input INPUT INPUT   Path of source and target corpus
  --output OUTPUT       Path to saved models
  --vocabulary VOCABULARY VOCABULARY
                        Path of source and target vocabulary
  --validation VALIDATION
                        Path of validation file
  --references REFERENCES [REFERENCES ...]
                        Path of reference files
  --model MODEL         Name of the model
  --parameters PARAMETERS
                        Additional hyper parameters
```

We distinguish between *required* and *optional* arguments. Users must specify the following required arguments to run the `trainer.py` script:

1. `--model`: the name of model architecture. It can be `seq2seq`, `rnnsearch` or `transformer`
2. `--input`: the path of source and target corpora.
3. `--vocabulary`: the path of source and target vocabularies.

The optional arguments of the `trainer.py` script can be omitted in a command. If an optional argument has a default value, the default value will be used in training if the argument is omitted in the command-line argument list. These optional arguments are listed as follows:

1. `--output`: the path to save checkpoints. The default value is `train` directory in the current path.
2. `--validation`: the path to the validation set. The best model will saved in `train/eval` directory.
3. `--references`: the path to references. Multiple references are also supported.

4. `--parameters`: other optional parameters which will be described in the next subsection. The parameters is a string containing comma-separated `name=value` pairs. Please refer to the document of `tf.contrib.training.parse_values`.

5. `--help`show this help message.

3.5 General Parameters

1. `num_threads`: the number of threads used in data processing.

2. `buffer_size`: the buffer size used in data processing.

3. `batch_size`: the batch size used in training stage.

4. `constant_batch_size`: if set to true, then each batch will contains `batch_size` sentences. Otherwise, each batch will contains approximately `batch_size` tokens.

5. `max_length`: the maximum length of training sentences.

6. `train_steps`: the total number of steps in the training stage.

7. `save_checkpoint_steps`: save the checkpoint every `N` steps.

8. `save_checkpoint_secs`: save the checkpoint every `N` second.

9. `initializer`: choose different initializing functions. The possible values are `uniform_unit_scaling`, `uniformnormal` and `normal_unit_scaling`.

10. `initializer_gain`: set the parameter of the initializer.

11. `learning_rate`: adjust the learning rate.

12. `learning_rate_decay`: set different learning rate decay function. The possible values are `noam`(used in the transformer architecture)`piecewise_constant`(see `tf.train.piecewise_constant`) and `none`.

13. `learning_rate_boundaries`: see `tf.train.piecewise_constant` for more references.

14. `learning_rate_values`: see `tf.trian.piecewise_constant` for more references.

15. `keep_checkpoint_max`: the maximum number of checkpoints to keep.

16. `keep_top_checkpoint_max`: the maximum number of top performed checkpoints to keep.

17. `eval_steps`: validate the model every `N` steps.

18. `eval_secs`: validate the model every `N` seconds.

19. `eval_batch_size`: the batch size used when validating.
20. `decode_alpha`: the length penalty term in the beam search. See (Wu et al., 2016) for more references.
21. `beam_size`: the beam size of beam search.
22. `decode_length`: the maximum length of decoded length, which is the length of source sentence plus this value.
23. `device_list`: the device id of GPUs. For example, set `device_list=[0,1]` will use GPU 0 and 1. Each GPU will process `batch_size` data.

3.6 Parameters specific to Seq2Seq

1. `rnn_cell`: the recurrent unit. Currently, only LSTM is supported.
2. `embedding_size`: the size of source and target embedding.
3. `hidden_size`: the hidden units of RNN layer.
4. `num_hidden_layers`: the number of RNN layers.
5. `dropout`: dropout rate used in the network.
6. `label_smoothing`: the value of label smoothing.
7. `reverse_source`: if set to true, reverse the source sentence automatically (Sutskever et al., 2014).
8. `use_residual`: whether to use residual connections when multi-layered LSTM is employed.

3.7 Parameters specific to RNNsearch

1. `rnn_cell`: the recurrent unit. Currently, only GRU is supported.
2. `embedding_size`: the size of source and target embedding.
3. `hidden_size`: the hidden units of RNN layer.
4. `maxnum`: the hidden units of maxout layer.
5. `dropout`: dropout rate used in the network.
6. `label_smoothing`: the value of label smoothing.

3.8 Parameters specific to Transformer

1. `hidden_size`: the embedding size and hidden size of the network.
2. `filter_size`: the hidden size of Feed-forward layer.
3. `num_encoder_layers`: the number of encoder layers.
4. `num_decoder_layers`: the number of decoder layers.
5. `num_heads`: the number of attention heads used in attention mechanism.
6. `shared_embedding_and_softmax_weights`: whether to share the embedding and softmax weights.
7. `shared_source_target_embedding`: whether to share the source and target embedding.
8. `residual_dropout`: the dropout rate used in residual connection.
9. `attention_dropout`: the dropout rate used in attention mechanism.
10. `relu_dropout`: the dropout rate used in feed forward layer.
11. `label_smoothing`: the value of label smoothing.

3.9 Test

3.9.1 The Python Script for Test

The `translator.py` in the `bin` folder is used to translate unseen test sentences. It has the following arguments:

```
usage: translator.py [<args>] [-h | --help]

Translate using existing NMT models

optional arguments:
  -h, --help            show this help message and exit
  --input INPUT         Path of input file
  --output OUTPUT       Path of output file
  --checkpoints CHECKPOINTS [CHECKPOINTS ...]
                        Path of trained models
  --vocabulary VOCABULARY VOCABULARY
                        Path of source and target vocabulary
  --models MODELS [MODELS ...]
                        Name of the model
  --parameters PARAMETERS
                        Additional hyper parameters
```

Users must specify the following required arguments to run the `translator.py` script:

1. `--input`: the path to input file.
2. `--output`: the path to translated results.
3. `--checkpoints`the path to trained checkpoints.
4. `--vocabulary`: the path to source and target vocabulary.
5. `--models`: the name of used architecture.

The optional argument of `translator.py` is listed below:

1. `--parameters`: Change parameters used in decoding. Such as `beam_size` or the GPU id.
2. `--help`: Show the help message.

3.9.2 Decoding

The source file of the test set `test.src` in the data folder is

```
ta xihuan huahua me ?
ta yidian dou bu xihuan huahua .
```

Given a directory to trained models `train`, please run the following command to translate the test set without evaluation:

```
python /PATH/TO/THUMT/thumt/bin/translator.py
  --models transformer
  --input test.src
  --output test.trans
  --checkpoints train
  --vocabulary vocab.zh vocab.en
  --parameters=device_list=[0]
```

Note that `test.trans` is the output of THUMT.

3.10 Model Averaging

Model averaging is also called checkpoint ensemble, which is used to average the checkpoints saved during training. The `checkpint_averaging.py` script is used to average checkpoints, which has the following arguments:

```
usage: average.py [<args>] [-h | --help]

Average checkpoints

optional arguments:
  -h, --help            show this help message and exit
  --path PATH           checkpoint dir
  --checkpoints CHECKPOINTS
```

```
                        number of checkpoints to use
  --output OUTPUT       output path
```

The meaning of arguments is shown below:

1. `path`: the path to checkpoints, which is the `output` argument specified during training.
2. `--checkpoints`: the number of checkpoints to average. It will load `N` latest checkpoints.
3. `--output`: the output directory.
4. `--help`show the help message.

3.11 Examples

The following is the command we used to train a base Transformer model. This model shares the source, target and softmax weights. We employed 4 GPUs to accelerate training:

```
python /PATH/TO/THUMT/thumt/bin/trainer.py
  --input train.tok.clean.bpe.32000.en.shuf
          train.tok.clean.bpe.32000.de.shuf
  --model transformer --output train/
  --vocabulary vocab.bpe.32000 vocab.bpe.32000
  --validation newstest2013.tok.bpe.32000.en
  --references newstest2013.tok.bpe.32000.de
  --parameters=batch_size=6250,device_list=[0,1,2,3],
               eval_steps=5000,train_steps=100000,
               save_checkpoint_steps=1500,
               shared_embedding_and_softmax_weights=true,
               shared_source_target_embedding=true
```

Once the training stage is finished, we use th following command to average checkpoints:

```
python /PATH/TO/THUMT/thumt/scripts/checkpoint_averaging.py
  --path train --checkpoints 5 --output average
```

Finally, we use the `translator.py` to translate the test set.

```
python /PATH/TO/THUMT/thumt/bin/translator.py
  --model transformer
  --checkpoints average
  --input newstest2014.tok.bpe.32000.en
  --vocabulary vocab.bpe.32000 vocab.bpe.32000
  --output output.txt
```

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*.

Chiang, D. (2005). A hierarchical phrase-based model for statistical 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*.

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. *arXiv preprint arXiv:1706.03762*.

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.