THUMT: An Open-Source Toolkit for Neural Machine Translation

The Tsinghua Natural Language Processing Group

June, 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 Theano, 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).

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.

3. *Exploiting monolingual data.* THUMT provides semi-supervised training (SST) for NMT (Cheng et al., 2016) that is capable of exploiting abundant monolingual corpora to improve the learning of both source-to-target and target-to-source NMT models.

4. *Visualization.* To better understand the internal workings of NMT, THUMT features a visualization tool to demonstrate the relevance between each intermediate state and its relevant contextual words (Ding et al., 2017).

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. FastAlign (optional, only used for replacing unknown words).
3. 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 Theano:

```
pip install argparse
pip install theano==0.8.2
```

Please visit https://github.com/clab/fast_align to download and install FastAlign.

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 Step 1: Unpacking

Unpack the package using the following command:

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

Entering the `THUMT` folder, you may find five folders (`thumt`, `scripts`, `config`, `data`, `viz`) and two files (`LICENSE` and `UserManual.pdf`):

1. `thumt`: the source code.
2. `scripts`: scripts for training and test.
3. `config`: an example configuration file for training.
4. `data`: toy training, validation, and test datasets.
5. `viz`: visualization code, GUI tool and example data.
6. `LICENSE`: license statement.
7. `UserManuel.pdf`: this document.

2.3.2 Step 2: Modifying Scripts

Running THUMT is mainly done by using three Python scripts in the `scripts` folder:

1. `trainer.py`: training translation models.
2. `test.py`: testing translation models.
3. `visualize.py`: visualizing the translation process.

You need to enable these scripts to locate other Python programs by modifying the `root_dir` variable in each script.

For the `trainer.py` script, you may change line 8

```
1 root_dir = '/data/disk1/private/ly/THUMT'
```

to

```
1 root_dir = '/User/Jack/THUMT'
```

where `'/Users/Jack/THUMT'` is an example folder where THUMT is installed. You may use the `pwd` command to get the full path of the root directory of THUMT on your own computer.

Similarly, you need to modify line 9 of the `test.py` script

```
1 root_dir = '/data/disk1/private/ly/THUMT'
```

to

```
1 root_dir = '/User/Jack/THUMT'
```

Line 6 of the `visualize.py` also needs to change from

```
1 root_dir = '/data/disk1/private/ly/THUMT'
```

to

```
1 root_dir = '/User/Jack/THUMT'
```

If you want to use techniques for addressing unknown words (Luong et al., 2015), please change line 10 of `mapping.py` in the `thumt` folder

```
1 aligner = ''
```

to

```
1 aligner = '/Users/Jack/fast_align/build/fast_align'
```

so as to locate the executable of FastAlign. Please use `pwd` to obtain the actual path on your own computer.

2.3.3 Step 3: Modifying Environment Variables

We highly recommend running THUMT on GPU servers. Suppose THUMT runs on NVIDIA GPUs with the CUDA toolkit version 7.5 installed. Users need to set environment variables to enable the GPU support:

```
export PATH=/usr/local/cuda-7.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
```

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.

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 .
```

[1] The Chinese text is romanized for readability.

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.

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 supports multiple references and uses a naming scheme slightly different from single-reference validation sets. Suppose there are four reference translations for `valid.src`. They should be named as `valid.trg0`, `valid.trg1`, `valid.trg2`, and `valid.trg3`. In other words, multiple references must be numbered starting from zero and share the same prefix.

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

3.2.1 The Python Script for Training

THUMT implements the standard attention-based encoder-decoder framework for neural machine translation (Bahdanau et al., 2015). It supports three training criteria: maximum likelihood estimation (MLE) (Bahdanau et al., 2015), minimum risk training (MRT) (Shen et al., 2016), and semi-supervised training (SST) (Cheng et al., 2016). The default training criterion is MLE, which is often used to initialize MRT and SST. We recommend using MRT for high-resource languages and SST for low-resource languages.

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

```
Usage: trainer [--help] ...
Required arguments:
  --config-file <file>           configuration file
  --trn-src-file <file>          training set, source file
  --trn-trg-file <file>          training set, target file
  --vld-src-file <file>          validation set, source file
  --vld-trg-file <file>          validation set, target file
  --device {cpu, gpu0, ...}      device
Optional arguments:
  --training-criterion {0,1,2}   training criterion
                                   0: MLE (default)
                                   1: MRT
                                   2: SST
  --replace-unk {0,1}            replacing unknown words
                                   0: off
                                   1: on (default)
  --save-all-models {0,1}        saving all intermediate models
                                   0: the best model (default)
                                   1: all intermediate models
  --mono-src-file <file>         monolingual source file
  --mono-trg-file <file>         monolingual target file
  --init-model-file <file>      initialization model file
  --debug {0,1}                  displaying debugging info
                                   0: off (default)
                                   1: on
  --help                         displaying this message
```

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

1. `--config-file`: specify the path to the configuration file (see Section 3.2.2 for details) that sets the values of vocabulary size, embedding dimension, mini-batch size, etc (e.g., `THUMT.config` in the `config` folder).

2. `--trn-src-file`: specify the source file of the training set (e.g., `train.src` in the `data` folder).

3. `--trn-trg-file`: specify the target file of the training set (e.g., `train.trg` in the `data` folder).

4. `--vld-src-file`: specify the source file of the validation set (e.g., `valid.src` in the `data` folder).

5. `--vld-trg-file`: specify the target file of the validation set (e.g., `valid.trg` in the toy data). If there are multiple target files such as `valid.trg0`, `valid.trg1`, `valid.trg2`, and `valid.trg3`, users need to set the value of this argument to the shared prefix `valid.trg`.

6. `--device`: specify the device for running this script. For NVIDIA GPUs, the `nvidia-smi` command can be used to find an unoccupied GPU. If no GPUs are available, you may set the value of this argument to "cpu" and run the `trainer.py` on CPU servers. Note that the training time on CPUs is usually orders of magnitude longer than on GPUs.

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. `--training-criterion`: specify the training criterion. Value 0 stands for maximum likelihood estimation (MLE) (Section 3.2.3) , 1 for minimum risk training (MRT) (Section 3.2.4), and 2 for semi-supervised training (SST) (Section 3.2.5). The default value of this argument is 0. In other words, the value of `--training-criterion` is set to 0 if this argument is not included in the command-line argument list.

2. `--replace-unk`: specify whether to address unknown words or not. Value 0 turns this option off and 1 turns it on. The default value is 1.

3. `--save-all-models`: specify whether to save all intermediate models during training. Value 0 only saves the best model and 1 saves all models. Note that saving all models often requires a large amount of disk space. The default value is 0.

4. `--mono-src-file`: specify the monolingual source file used for SST (Section 3.2.5).

5. `--mono-trg-file`: specify the monolingual target file used for SST (Section 3.2.5).

6. `--debug`: specify whether to display debugging information. Value 0 stands for turning this option off and 1 for on. The default value is 0.

7. `--help`: displaying helping message.

   In Sections 3.2.3-3.2.5, we will introduce how to use this script in detail.

3.2.2 Configuration File

Although NMT is capable of learning expressive representations from data, its translation performance heavily depends on the setting of hyper-parameters such as vocabulary size, layer dimension, mini-batch size, and learning rate.

Instead of using command-line arguments, THUMT uses a configuration file to specify the setting of hyper-parameters. In the `config` folder, there is an example configuration file `THUMT.config`:

```
### vocabulary size ###
# source vocabulary size: [1, +00)
[source vocabulary size] 30000
# target vocabulary size: [1, +00)
[target vocabulary size] 30000

### network architecture ###
# source word embedding dimension: [1, +00)
[source word embedding dimension] 620
# target word embedding dimension: [1, +00)
[target word embedding dimension] 620
# encoder hidden layer dimension: [1, +00)
[encoder hidden layer dimension] 1000
# decoder hidden layer dimension: [1, +00)
[decoder hidden layer dimension] 1000
# dropout: [0,1)
[dropout ratio] 0.2

### training ###
## minimum risk training (MRT) setting
# number of sentences to be sampled: [1, +00)
[MRT sample size] 100
# length ratio limit of sampled sentences: (0, +00)
[MRT length ratio limit] 1.5

### training ###
# maximum sentence length: [1, +00)
[maximum sentence length] 50
# number of sentences in a mini-bath: [1, +00).
[mini-batch size] 80
# number of mini-batches to be sorted: [1, +00)
[mini-batch sorting size] 20
# iteration limit: [1, +00)
[iteration limit] 1000000
# convergence limit: [1, +00)
[convergence limit] 100000

### optimization ###
```

```
# optimizer: 0 for SGD, 1 for AdaDelta, 2 for Adam
[optimizer] 2
# gradient clipping
[clip] 1.0

# SGD setting
[SGD learning rate] 1.0

# AdaDelta setting
[AdaDelta rho] 0.95
[AdaDelta epsilon] 1e-6

# Adam setting
[Adam alpha] 0.0005
[Adam alpha recover] 0.998
[Adam beta1] 0.9
[Adam beta2] 0.999
[Adam epsilon] 1e-8
[Adam decay] 0.5

### search ###
[beam size] 10

### model dumping ###
# interval for dumping and validating intermediate models
[model dumping iteration] 10000
# interval for saving checkpoints
[checkpoint iteration] 2000
```

Here is a list of hyper-parameters used in THUMT:

1. `[source vocabulary size]`: the size of source language vocabulary. Due to the memory limit of GPUs as well as computation efficiency, only most frequent words are included in the vocabulary. Larger vocabulary sizes often lead to better translation quality but increase memory requirements and computation costs.
2. `[target vocabulary size]`: the size of target language vocabulary.
3. `[source word embedding dimension]`: the dimension of source word embedding. Large dimensions improve expressive power but increase memory requirements and computation costs.
4. `[target word embedding dimension]`: the dimension of target word embedding.
5. `[encoder hidden layer dimension]`: the dimension of encoder hidden layer. Large dimensions improve expressive power but increase memory requirements and computation costs.

6. `[decoder hidden layer dimension]`: the dimension of decoder hidden layer.

7. `[dropout ratio]`: the probability of dropout.

8. `[MRT sample size]`: number of sentences to be sampled for minimum risk training (Shen et al., 2016). Larger sizes often lead to better translation quality. Please carefully choose an appropriate size to avoid exceeding the memory limit of GPUs.

9. `[MRT length ratio limit]`: the length ratio limit of sampled sentences. This hyper-parameter prevents the sampler from choosing too long candidate translations.

10. `[maximum sentence length]`: length limit of sentences in the training set.

11. `[mini-batch size]`: the number of sentences in a mini-batch. Due to the memory limit, this hyper-parameter MUST be set to 1 for MRT.

12. `[mini-batch sorting size]`: the number of mini-batches to be sorted.

13. `[iteration limit]`: the limit of iterations in training. The training ends when the iteration reaches its limit. For example, if this hyper-parameter is set to 1,000,000, the `trainer.py` script runs for at most 1,000,000 iterations.

14. `[convergence limit]`: the number of iterations without increasing BLEUs before convergence. For example, if this hyper-parameter is set to 100,000, the training process will be terminated if the highest BLEU score on the validation set has not changed for 100,000 iterations.

15. `[optimizer]`: THUMT supports three optimization methods. Value 0 stands for stochastic gradient descent (SGD), 1 for AdaDelta (Zeiler, 2012), 2 for Adam (Kingma and Ba, 2014).[2] We recommend using Adam for THUMT.

16. `[clip]`: gradient clipping for addressing the gradient explosion problem.

17. `[SGD learning rate]`: learning rate in SGD.

18. `[AdaDelta rho]`: decay rate in AdaDelta.

19. `[AdaDelta epsilon]`: a constant used to better condition the denominator in AdaDelta.

20. `[Adam alpha]`: step size in Adam. We recommend setting this hyper-parameter to 0.0005 for MLE, 0.00001 for MRT, and 0.00005 for SST.

[2] We implement Adam in a slightly different way to avoid the occurrence of NaN during training.

21. `[Adam alpha recover]`: the exponential recover rate for step size in Adam.

22. `[Adam beta 1]`: exponential decay rate for the moment estimates in Adam.

23. `[Adam beta 2]`: another exponential decay rate for the moment estimates in Adam.

24. `[Adam epsilon]`: a small constant in Adam.

25. `[Adam decay]`: the decay rate for step size in Adam after each epoch.

26. `[beam size]`: the beam size in decoding.

27. `[model dumping iteration]`: interval for dumping and validating intermediate models. If this hyper-parameter is set to 10,000, the trainer will dump an intermediate model and evaluate it on the validation set every 10,000 iterations.

28. `[checkpoint iteration]`: interval for saving checkpoints that are used to resume training when interrupted. If this hyper-parameter is set to 2,000, the trainer will save checkpoints every 2,000 iterations.

3.2.3 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is the default training criterion in THUMT. The model trained using MLE is often used to initialize MRT and SST.

To train NMT models using the MLE criterion, run the following one-line command:

```
python /User/Jack/THUMT/scripts/trainer.py --config-file /Users/
Jack/THUMT/data/THUMT_toy.config --trn-src-file /Users/Jack/
THUMT/data/train.src --trn-trg-file /Users/Jack/THUMT/data/
train.trg --vld-src-file /Users/Jack/THUMT/data/valid.src
--vld-trg-file /Users/Jack/THUMT/data/valid.trg --device gpu0
```

Note that we suppose `gpu0` is available. In practice, please use the `nvidia-smi` command to find an unoccupied GPU. We use `THUMT_toy.config` in the `data` folder, which is more suitable for running THUMT on toy data than the standard configuration file `THUMT.config` in the `config` folder.

For large training sets that contain millions of sentence pairs, it usually takes THUMT several days to converge, depending on the `[iteration limit]` and `[convergence limit]` hyper-parameters in the configuration file. To monitor the training process, the `trainer.py` script dumps the results of validation to a file called `log`. Here is an example `log` file:

```

               Time     Iteration        Cost        BLEU
-------------------------------------------------------------
2017-05-05 03:39:39          2000     5804.74       25.00
```

```
2017-05-05 04:21:01          4000      5288.56       27.49
2017-05-05 05:02:04          6000      4905.22       29.59
2017-05-05 05:43:14          8000      4651.91       30.28
2017-05-05 06:24:22         10000      4474.42       31.81

```

The `log` file records the validation time, the iteration, the average cost, and the BLEU score for each dumped intermediate model.

Besides the `log` file, THUMT also generates and updates checkpoint files `checkpoint_config.pkl`, `checkpoint_model.npz`, `checkpoint_status.pkl`, which are used for resuming training if the training process is interrupted. The interval for saving checkpoints is determined by `[checkpoint iteration]` in the configuration file. To resume the training from checkpoints, users only need to re-run the original command.

If the `--save-all-models` option of the `trainer.py` script is turned on (i.e., its value is 1), THUMT will create a folder `models` to store intermediate models, which are named according to the iterations at which they are dumped. For example, the model dumped at iteration 2,000 is named as `model_iter2000.npz`. As saving intermediate models usually leads to large disk space requirements, the default setting of the `trainer.py` script is to turn the `--save-all-models` option off. THUMT also creates a folder `corpus` to store pre-processed training data and another folder `valid` to store the translation result file for each intermediate model.

During training, THUMT maintains the best model `model_best.npz`, which is the intermediate model that achieves the highest BLEU score on the validation set. After the MLE training is done, the resulting model file can be re-named as

```
mv model_best.npz model_mle_best.npz
```

3.2.4 Minimum Risk Training

Minimum risk training (MRT) proves to significantly and consistently improve translation quality over MLE (Shen et al., 2016).

Before starting MRT, it is important to first set the `[mini-batch size]` hyper-parameter of the `THUMT.config` file to 1 due to memory limit. We recommend using Adam as the optimizer and setting `[Adam alpha]` to 0.00001. As MRT with random initialization often takes a long time, we recommend initializing MRT with the best model output by MLE using the `--init-model-file` option.

The command for running MRT is given by

```
python /Users/Jack/THUMT/scripts/trainer.py --config-file /Users/
Jack/THUMT/config/THUMT.config --trn-src-file /Users/Jack/THUMT/
data/train.src --trn-trg-file /Users/Jack/THUMT/data/train.
trg --vld-src-file /Users/Jack/THUMT/data/valid.src --vld-trg-
file /Users/Jack/THUMT/data/valid.trg --training-criterion 1
```

```
--init-model-file model_mle_best.npz --device gpu0
```

After the training is done, the resulting model file can be re-named as

```
mv model_best.npz model_mrt_best.npz
```

Currently, THUMT only supports running MRT on single GPUs. In the original MRT paper (Shen et al., 2016), multiple GPUs are used so that more samples can fit in the memory and improve translation quality. THUMT supporting multiple GPUs will be released in the future.

3.2.5 Semi-Supervised Training

Semi-supervised training (SST) is capable of exploiting monolingual corpora to improve bi-directional NMT (Cheng et al., 2016). This is very useful for low-resource language translation without abundant bilingual corpora available.

SST assumes that the following models and corpora are available:

1. `model_s2t_init.npz`: source-to-target translation model.
2. `model_t2s_init.npz`: target-to-source translation model.
3. `mono.zh`: source-language monolingual corpus.
4. `mono.en`: target-language monolingual corpus.

Often, we use MLE to obtain `model_s2t_init.npz` and `model_t2s_init.npz`, which serve as the starting point of SST. First, the two models need to be merged into one model:

```
python /Users/Jack/THUMT/thumt/merge_semi.py model_s2t_init.npz
model_t2s_init.npz model_semi_init.npz
```

The monolingual source file `mono.src` in the `data` folder is

```
wo yidian dou bu xihuan yinyue .
wo ye hen xihuan huahua .
ta xihuan yinyue me ?
```

The monolingual target file `mono.trg` in the `data` folder is

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

Note that the two monolingual corpora are only loosely related and do not constitute a parallel corpus.

As SST approximates the full search space with a small set of candidate translations, it is necessary to choose an appropriate value of `[mini-batch size]` to avoid exceeding GPU memory limit. Therefore, users need to change the `[mini-batch size]` hyper-parameter of the configuration file `THUMT.config` to 3.

The command for running SST is shown as follows:

```
python /Users/Jack/THUMT/scripts/trainer.py --config-file /Users/
Jack/THUMT/data/THUMT_toy.config --trn-src-file /Users/Jack/
THUMT/data/train.src --trn-trg-file /Users/Jack/THUMT/data/
train.trg --vld-src-file /Users/Jack/THUMT/data/valid.src
--vld-trg-file /Users/Jack/THUMT/data/valid.trg --training
-criterion 2 --mono-src-file /Users/Jack/THUMT/data/mono.src
--mono-trg-file /Users/Jack/THUMT/data/mono.trg --init-model
-file model_semi_init.npz --device gpu0
```

As SST jointly trains source-to-target and target-to-source models on four corpora, it usually requires a much longer time than MLE to converge. After the training is done, the resulting model file can be split into trained source-to-target and target-to-source models using `split_semi.py` in the `thumt` folder:

```
python /Users/Jack/THUMT/thumt/split_semi.py model_best.npz model
_s2t_best.npz model_t2s_best.npz
```

3.3 Test

3.3.1 The Python Script for Test

The `test.py` script in the `scripts` folder is used to translate unseen sentences of a test set and report BLEU scores. Its arguments are listed as follows:

```
Usage: test.py [--help] ...
Required arguments:
  --model-file <file>        model file(s)
  --test-src-file <file>     test set, source file
  --test-trg-file <file>     translation of the test set
  --device {cpu,gpu0,...}    the device for running this script
Optional arguments:
  --test-ref-file <file>     test set, reference file(s)
  --replace-unk              replacing unknown words
                                0: off
                                1: on (default)
  --help                     displaying helping message
```

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

1. `--model-file`: the trained model file. Input multiple models for ensemble.
2. `--test-src-file`: the source file of the test set.
3. `--test-trg-file`: the translation of the test set output by THUMT.

4. `--device`: the device for running this script.

The optional arguments of the `test.py` script are

1. `--test-ref-file`: the reference file(s) of the test set.
2. `--replace-unk`: replacing unknown words in post-processing. Please keep the consistency of the `--replace-unk` options between training and test. For example, if `--replace-unk` is turned off in the `training.py` script, we recommend turning this option off in the `test.py` script too.
3. `--help`: displaying helping message.

3.3.2 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 trained model `model_best.npz`, please run the following command to translate the test set without evaluation:

```
python /Users/Jack/THUMT/scripts/test.py --model-file model_best.
npz --test-src-file /Users/Jack/THUMT/data/test.src --test-trg
-file test.trans --device gpu0
```

Note that `test.trans` is the translation of the test set output by THUMT.

3.3.3 Evaluation

Provided with reference translations, either single (e.g., `test.trg`) or multiple (e.g., `test.trg0`, `test.trg1`, ...), the `test.py` can also be used for calculating BLEU scores on the test set. There is a single-reference target file `test.trg` in the `data` folder:

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

The command for decoding and evaluation is given by

```
python /Users/Jack/THUMT/scripts/test.py --model-file model_best.
npz --test-src-file /Users/Jack/THUMT/data/test.src --test-trg
-file test.trans --device gpu0 --test-ref-file test.trg
```

The evaluation result will be dumped to a file `evalResult`.

3.4 Visualization

THUMT features a visualization tool for visualizing the internal workings of attention-based NMT (Ding et al., 2017). First, please use the `visualize.py` script in the `scripts` folder to calculate relevance between hidden states and context. Its arguments are listed as follows:

```
Usage: visualize.py [--help] ...
Required arguments:
  --model-file <file>          model file
  --input-file <file>          input source file
  --output-dir <dir>           output directory
  --device {cpu,gpu0,...}      the device for running this script
Optional arguments:
  --help                       displaying helping message
```

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

1. `--model-file`: the trained model file.
2. `--input-file`: the source file of the test set.
3. `--output-dir`: the directory for storing relevance files.
4. `--device`: the device for running this script.

An example command is given as follows.

```
python /Users/Jack/THUMT/scripts/visualize.py --model-file model_
best.npz --input-file /Users/Jack/THUMT/data/test.src --output
-dir relevance --device gpu0
```

After the calculation is done, the resulting files are zipped into `relevance.zip`, which is the input of the GUI tool THUMT-Viz. Click "`File >> Open ...`" to open the relevance file `relevance.zip`. Figure 1 shows an example. The GUI displays the neural network for a source sentence and its translation. There are eight types of layers: (1) source word embedding, (2) source forward hidden states, (3) source backward hidden states, (4) source concatenated hidden states, (5) attention, (6) source context, (7) target hidden states, and (8) target word embedding.

Clicking a node of the neural network, users will find the relevance between the node and its source and target contextual words that have an influence on its generation in the bottom area. The original relevance is calculated by decomposing the value of the node among all contextual words. Please refer to (Ding et al., 2017) for more details. To facilitate visualization, we normalize relevance among all source and target contextual words of the node.

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*.

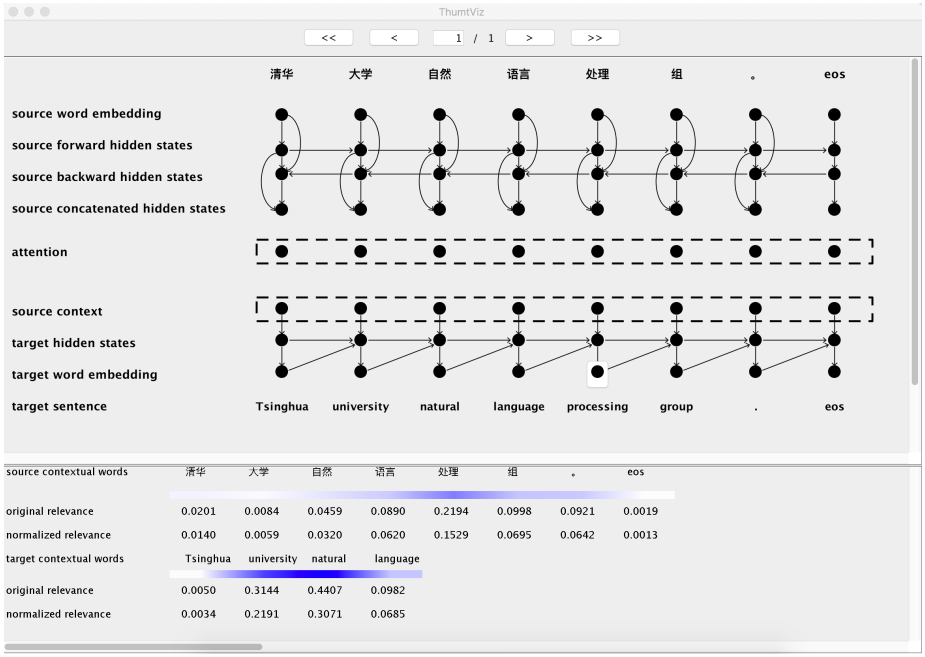

Figure 1: A GUI tool for visualizing attention-based neural machine translation.

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 hirrarchical 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*.

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

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

Luong, T., Sutskever, I., Le, Q., Vinyals, O., and Zaremba, W. (2015). Addressing the rare word problem in neural machine translation. In *Proceedings of ACL*.

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*.

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.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. arXiv:1212.5701v1.