THUMT: 开源神经机器翻译工具包

清华大学自然语言处理与人文计算实验室

2017 年 12 月

1 简介

机器翻译（Machine Translation，MT）是使用计算机来自动进行人类语言翻译，它是自然语言处理以及人工智能的一个重要任务。自 1990 年以来，随着平行语料的增长，数据驱动的机器翻译方法变得流行起来。近年来，端到端的神经机器翻译（Neural Machine Translation，NMT）(Sutskever et al., 2014; Bahdanau et al., 2015) 进展迅速。由于 NMT 能够从数据中学习到特征表示，NMT 迅速地取代了传统的统计机器翻译方法（Statistical Machine Translation，SMT）(Brown et al., 1993; Koehn et al., 2003; Chiang, 2005)，成为了实用 MT 系统中新的事实上的标准 (Wu et al., 2016)。

基于TensorFlow, THUMT 是一套开源的神经机器翻译工具包，由清华大学自然语言处理与人文实验室开发。THUMT 具有以下特性:

1. 多神经机器翻译架构。THUMT 实现了传统的序列到序列架构 (Sutskever et al., 2014)，标准的基于注意力机制的编码器-解码器架构 (Bahdanau et al., 2015) 以及基于自注意力机制的 Transfromer 架构 (Vaswani et al., 2017)。

2. 最小错误率训练。除了标准的最大似然估计（Maximum Likelihood Estimation，MLE），THUMT 也支持最小错误率训练（Minimum Risk Training，MRT）(Shen et al., 2016)。最小错误率训练旨在寻找能够最小化错误期望的参数，而错误期望通过 BLEU(Papineni et al., 2002) 等评测目标在训练集上计算。

2 安装

2.1 系统要求

THUMT 支持 Linux i686 以及 MacOS X。必须安装以下第三方软件以支持 THUMT:

1. Python 版本 2.7.0 或以上。

2. JRE 1.6 或以上 (可选，仅当可视化时需要)。

2.2 安装前提

我们推荐使用 pip 来安装 THUMT 的依赖。安装从 python-pip 开始：

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

然后，使用以下两个命令来安装 argparse 以及 TensorFlow（要求版本 >=1.4.0）：

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

2.3 安装 THUMT

THUMT 的源代码可以从工具网站(稳定版本) 以及 GitHub(最新版本) 中获得。以下是安装 THUMT 的简单指南。

2.3.1 第一步：解压

使用以下命令进行解压：

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

进入THUMT 目录，可以找到两个子目录 (thumt，和data) 以及三个文件 (LICENSE，UserManual.en.pdf 和UserManual.zh.pdf)：

1. thumt: 源代码。
2. data: 训练、验证以及测试集的玩具样例。
3. docs: 文档源文件。
4. LICENSE: 许可文件。
5. UserManual.en.pdf：英文文档。
6. UserManual.zh.pdf: 本文档。

2.3.2 第二步：更改环境变量

我们极力推荐在 GPU 服务器上运行 THUMT。假定 THUMT 在 NVIDIA GPUs 运行并且安装了CUDA 工具包版本 8.0，用户需要设置以下环境变量来使用 THUMT 以及 GPU：

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

若希望永久设置这些环境变量，用户可以将以上几行添加到$HOME 目录的.bashrc 文件中。其中/PATH/TO/THUMT 是 THUMT 所在目录。

3 使用手册

3.1 数据准备

运行 THUMT 会涉及到三种数据集：

1. 训练集：用于训练 NMT 的平行句对集合。
2. 验证集：由源端句子以及多个目标端翻译组成的数据集合，用于模型选择以及超参调节。
3. 测试集：由源端句子以及多个目标端翻译组成的数据集合，用于测试模型在未知数据上的表现。

3.1.1 训练集

训练集的作用是来训练 NMT 模型。它一般由两个文件组成：一个保存源端句子而另一个保存对应的目标端句子。在`data` 目录下，又一个名为`train.src` 的例示文件，它包含以下 7 句话：

```
我 很 喜欢 音乐 。
我 不 喜欢 画画 。
你 喜欢 音乐 么 ？
是的 ， 我 也 喜欢 音乐 。
他 也 喜欢 音乐 。
她 一点 都 不 喜欢 音乐 。
她 很 喜欢 画画 。
```

如下所示是对应的目标文件`train.trg` 的内容：

```
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 .
```

注意源端和目标端文件中的每一行之包含一句分词或 token 后的句子。相同行号的源句子和目标句子互为翻译。

我们的玩具训练集只包含 7 个句子。在实践中，NMT 通常需要数百万的平行句对才能获得较好的翻译结果。

3.1.2 验证集

验证集用来模型选择以及超参调整。在训练过程中，THUMT 每隔一段时间在验证集上进行模型验证。在验证集上获得最高 BLEU 的模型将会被保存起来。

验证集包含一个源文件以及一个或多个目标文件。在`data` 目录下有一个例示源文件`valid.src`，内容如下：

```
我 很 喜欢 画画 。
```

```
我 不 喜欢 音乐 。
```

我们的例示验证集之包含一个目标文件`valid.trg`，它包含文件`valid.src`的翻译：

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

由于自然语言表达非常丰富，一个源端句子一般对应多个参考译文。THUMT也支持使用多个参考译文。

3.1.3 测试集

测试集用来评估训练好的 NMT 模型在未见数据上的表现。与验证集类似，测试集也包好一个源文件一个一个或多个目标文件。

在`data` 目录下，有一个源测试文件`test.src`:

```
她 喜欢 画画 么 ？
他 一点 都 不 喜欢 画画 。
```

对应的目标文件`test.trg` 内容如下：

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

在我们的例子中，验证集和测试集只包含两个句子。在时间中，验证集和测试集通常包含数千个句子。

3.2 训练

3.3 数据处理

为了解决 NMT 所存在的词表限制问题，最常用的方式是使用 Byte Pair Encoding (BPE) 方法。BPE 的源代码可以在Subword-NMT中找到。

为了使用 BPE 来编码训练集，首先需要生成 BPE 操作文件。以下命令将会生成一个名为`bpe32k` 的文件，它包含 32K 个 BPE 操作。该命令同时生成两个名为`vocab.src` 和`vocab.trg` 的文件。

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

在生成 BPE 操作文件以及词表以后，需要对训练集、验证集以及测试集源端进行编码：

```
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
```

以下命令可以将 BPE 编码后的文件进行还原：

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

3.4 数据随机化

将训练数据随机打乱顺序对训练模型很有帮助。为此我们提供了`shuffle_corpus.py`脚本对数据进行打乱，它具有以下参数：

```
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
```

各个参数的含义如下：

1. `corpus`：源端以及目标端语料库。
2. `--suffix`：输出文件的后缀，默认`.shuf`。
3. `--seed`：随机数种子。
4. `--help`：显示帮助信息。

3.4.1 词表生成

在训练之前，首先要根据语料库构建词表。`scripts` 下的`build_vocab.py` 可以完成这个功能。它具有以下参数：

```
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.
```

各个参数的含义如下：

1. `corpus`：源端或目标端语料库。
2. `--references`：输出词表名。
3. `--limit`：词表限制大小。若不提供则使用语料库中所有出现的词。
4. `--control`：控制字符，以逗号隔开如 "`<pad>,<eos>,<unk>`"，控制字符将添加到词表的最前端。
5. `--help`：显示帮助信息。

3.4.2 训练所使用的 Python 脚本

THUMT 实现了传统的序列到序列架构 Seq2Seq(Sutskever et al., 2014)，标准的基于注意力机制的编码器-解码器架构 RNNsearch(Bahdanau et al., 2015) 以及基于自注意力机制的 Transfromer 架构 (Vaswani et al., 2017)。

`bin` 目录下的`trainer.py` 脚本用于训练 NMT 模型。它有以下参数：

```
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
```

我们区分必须参数以及可选参数。用户必须指定以下的参数来运行训练脚本`trainer.py`：

1. `--model`：所选择的架构，目前支持三种架构。可选择的值为`seq2seq`，`rnnsearch` 以及`transformer`。
2. `--input`：首次运行时需要指定，表示源端与目标端训练文件。
3. `--vocabulary`：首次运行时需要指定，表示源端与目标端词表所在路径。

`trainer.py` 脚本中的可选参数可以在命令中忽略，此时将使用默认参数。可选的参数如下：

1. `--output`：输出模型所在目录，默认为`train` 目录。
2. `--validation`：验证集源端文件所在路径。验证最优模型存储在输出目录的`eval` 子目录中。
3. `--references`：验证集参考译文所在路径。支持多个参考译文。
4. `--parameters`：指定其他可选参数，这些参数在下一小节中描述。可选参数按照`name=value` 的方式给出，并且多个参数间用逗号进行分隔。具体可以参考`tf.contrib.training.parse_values` 的文档。
5. `--help`：显示帮助信息。

3.5 通用参数

1. `num_threads`: 数据处理时使用的线程数目。
2. `buffer_size`: 数据处理时的缓冲大小。
3. `batch_size`: 训练时批量的大小。
4. `constant_batch_size`: 是否采用固定的批量大小。如果为 True，则输入数据包含`batch_size` 个数个句子。若为 False，则输入数据中大致含有`batch_size` 个词。
5. `max_length`: 限制训练数据中句子的最大长度。
6. `train_steps`: 训练的总步数。
7. `save_checkpoint_steps`: 按步设置保存频率，与`save_checkpoint_secs` 互斥。
8. `save_checkpoint_secs`: 按秒设置保存频率，与`save_checkpoint_steps` 互斥。

9. initializer: 使用不同的初始化函数。可选值为uniform_unit_scaling, uniform，normal 以及normal_unit_scaling。
10. initializer_gain: 设置初始化的范围。不同的输出化函数会产生不同的作用。
11. learning_rate: 设置初始学习率。
12. learning_rate_decay: 设置学习率衰减函数。可选值为noam(Transformer 架构的学习率衰减方法)、piecewise_constant(按照区间进行衰减) 以及none（不使用衰减）。
13. learning_rate_boundaries: 参考tf.train.piecewise_constant。设置学习率减小的边界。
14. learning_rate_values: 参考tf.trian.piecewise_constant。设置不同区间学习率的值。
15. keep_checkpoint_max: 最大保留的模型存储的个数。
16. keep_top_checkpoint_max: 保留最好模型的个数。
17. eval_steps: 每隔eval_steps 步验证一次模型。与eval_secs 互斥。
18. eval_secs: 每隔eval_secs 秒验证一次模型。与eval_steps 互斥。
19. eval_batch_size: 验证时采用的句子批量大小。
20. decode_alpha: 解码时长度惩罚，见 (Wu et al., 2016) 中的解码公式。
21. beam_size: Beam Search 时的 Beam 大小。
22. decode_length: 通过源端长度加decode_length 计算解码句子最大的长度。
23. decode_constant: 解码时长度惩罚中的常量，见 (Wu et al., 2016) 中的解码公式。
24. device_list: 使用 GPU 的编号，支持多 GPU 并行计算。例如，可以使用device_list=[0,1] 选择使用第 0 和 1 号 GPU。此时每个 GPU 会并行处理batch_size 的数据。

3.6 Seq2Seq 参数

1. rnn_cell: 使用的 RNN 单元。目前仅支持 LSTM。
2. embedding_size: 词向量大小。
3. hidden_size: RNN 隐层大小。
4. num_hidden_layers: RNN 层数。
5. dropout: Dropout 大小。

6. `label_smoothing`: 标签平滑大小。
7. `reverse_source`: 是否将源端句子反转，见 (Sutskever et al., 2014)。
8. `use_residual`: 多层 RNN 时是否使用残差连接。

3.7 RNNsearch 参数

1. `rnn_cell`: 使用的 RNN 单元。目前仅支持 GRU。
2. `embedding_size`: 词向量大小。
3. `hidden_size`: RNN 隐层大小。
4. `maxnum`: Maxout 层的参数。
5. `dropout`: Dropout 大小。
6. `label_smoothing`: 标签平滑大小。

3.8 Transformer 参数

1. `hidden_size`: 网络中词向量以及每层输出的大小。
2. `filter_size`: 前馈层隐层的大小。
3. `num_encoder_layers`: 编码器层数。
4. `num_decoder_layers`: 解码器层数。
5. `num_heads`: Multi-head 注意力机制中 head 数目。
6. `shared_embedding_and_softmax_weights`: 共享 Softmax 层以及目标端词向量的参数。
7. `shared_source_target_embedding`: 共享源端与目标端词向量参数。
8. `residual_dropout`: 残差连接的 Dropout 大小。
9. `attention_dropout`: 注意力机制中 Dropout 大小。
10. `relu_dropout`: 前馈层中隐层 Dropout 大小。
11. `label_smoothing`: 标签平滑大小。

3.9 测试

3.9.1 测试使用的 Python 脚本

bin 目录下的translator.py 脚本用来翻译测试集中的句子。它有以下的参数可以指定：

```
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
```

用户必须指定以下的参数才可以运行translator.py

1. --input：需要进行翻译的文件。
2. --output：输出的文件名。
3. --checkpoints：训练好的模型所在的目录，支持多个模型集成。
4. --vocabulary：源端与目标端的词表位置。
5. --models：所使用的架构名称，支持不同架构之间的集成。

translator.py 的可选参数如下：

1. --parameters：调整解码中的其他参数。比如beam_size 以及 GPU 编号等。
2. --help：显示帮助信息。

3.9.2 解码

data 目录下的源端测试文件test.src 内容如下：

```
她 喜欢 画画 么 ?
他 一点 都 不 喜欢 画画 。
```

给定一个训练好的模型目录train，可以使用以下的命令来进行翻译：

```
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]
```

注意test.trans 是 THUMT 输出的翻译结果。

3.10 模型平均

模型平均也成为存储点集成 (checkpoint ensemble)，用于平均一次训练中存储的多个模型的参数。我们提供checkpint_averaging.py 脚本来完成这个步骤，它具有以下的参数：

```
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
```

各个参数的含义如下：

1. path：存储点所在的目录。通常是训练时指定的目录。
2. --checkpoints：需要平均的存储点数目。脚本将自动读取最近的N 个模型进行平均。
3. --output：输出的目录。在解码时指定该目录即可使用平均后的模型。
4. --help：显示帮助信息。

3.11 运行例示

以下是训练 Transformer 模型的命令。该模型共享源端、目标端以及 softmax 参数，使用 4 块 GPU 进行训练：

```
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
```

训练完成以后，使用以下命令进行模型平均：

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

以下是解码 Transformer 模型的命令：

```
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
```

参考文献

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.