

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 11 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：

```
1 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 $\geq 1.4.0$ ）：

```
1 pip install argparse
2 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：

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

进入 THUMT 目录，可以找到三个子目录 (thumt, config 和 data) 以及两个文件 (LICENSE 和 UserManual.pdf)：

1. thumt: 源代码。
2. config: 训练的配置文件。
3. data: 训练、验证以及测试集的玩具样例。
4. LICENSE: 许可文件。
5. UserManual.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 the CUDA toolkit 版本 8.0，用户需要设置以下环境变量来使用 THUMT 以及 GPU：

```
1 export PATH=/usr/local/cuda/bin:$PATH
2 export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
3 export PYTHON_PATH=/PATH/TO/THUMT:$PYTHON_PATH
```

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

训练集的作用是用来训练 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` 的示例文件，它包含以下 7 句话：

```
1 我 很 喜 欢 音 乐 。
2 我 不 喜 欢 画 画 。
3 你 喜 欢 音 乐 么 ？
4 是 的 ， 我 也 喜 欢 音 乐 。
5 他 也 喜 欢 音 乐 。
6 她 一 点 都 不 喜 欢 音 乐 。
7 她 很 喜 欢 画 画 。
```

如下所示是对应的目标文件 `train.trg` 的内容：

```
1 i like music very much .
2 i do not like painting .
3 do you like music ?
4 yes , i like music too .
5 he also likes music .
6 she does not like music at all .
7 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` 目录下有一个示例源文件 `valid.src`，内容如下：

```
1 我 很 喜 欢 画 画 。
```

```
2 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：

```
1 i like painting very much .
2 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或多个目标文件。

在`data` 目录下，有一个源测试文件`test.src`:

```
1 她 喜欢 画画 么 ？
2 他 一点 都 不 喜欢 画画 。
```

对应的目标文件`test.trg` 内容如下：

```
1 does she like painting ?
2 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

3.2.1 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：

```
1 usage: build_vocab.py [-h] [--limit LIMIT] [--control CONTROL]
2                        corpus output
3
4 Create vocabulary
5
6 positional arguments:
7   corpus                input corpus
8   output                Output vocabulary name
9
10 optional arguments:
11   -h, --help            show this help message and exit
12   --limit LIMIT         Vocabulary size
13   --control CONTROL     Add control symbols to vocabulary.
14                        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。

3.2.2 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)。

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

```

1 usage: trainer.py [<args>] [-h | --help]
2
3 Training neural machine translation models
4
5 optional arguments:
6   -h, --help            show this help message and exit
7   --input INPUT INPUT  Path of source and target corpus
8   --output OUTPUT      Path to saved models
9   --vocabulary VOCABULARY VOCABULARY
10                        Path of source and target vocabulary
11   --validation VALIDATION
12                        Path of validation file
13   --references REFERENCES [REFERENCES ...]
14                        Path of reference files
15   --model MODEL        Name of the model
16   --parameters PARAMETERS
17                        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。

3.3 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.train.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。

3.4 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。

3.5 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。

3.6 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。

3.7 测试

3.7.1 测试使用的 Python 脚本

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

```
1 usage: translator.py [<args>] [-h | --help]
2
3 Translate using existing NMT models
4
5 optional arguments:
6   -h, --help            show this help message and exit
7   --input INPUT          Path of input file
8   --output OUTPUT        Path of output file
9   --checkpoints CHECKPOINTS [CHECKPOINTS ...]
10                          Path of trained models
11   --vocabulary VOCABULARY VOCABULARY
12                          Path of source and target vocabulary
13   --models MODELS [MODELS ...]
14                          Name of the model
15   --parameters PARAMETERS
16                          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。

3.7.2 解码

`data` 目录下的源端测试文件`test.src` 内容如下:

```
1 她 喜欢 画画 么 ?  
2 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:

```
1 python /PATH/TO/THUMT/thumt/launcher/translator.py  
2 --models transformer  
3 --input test.src  
4 --output test.trans  
5 --checkpoints train  
6 --vocabulary vocab.zh vocab.en  
7 --parameters=device_list=[0]
```

注意`test.trans` 是 THUMT 输出的翻译结果。

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*.