Thot Toolkit for Statistical Machine Translation

Quick User Guide

Daniel Ortiz Martínez
daniel.o@webinterpret.com
Webinterpret

March 2017

1 Installation

1.1 Basic Installation Procedure

The code of the Thot toolkit is hosted on github[a]. To install Thot, first you need to install the autotools (autoconf, autoconf-archive, automake and libtool packages in Ubuntu). If you are planning to use Thot on a Windows platform, you also need to install the Cygwin environment[b]. Alternatively, Thot can also be installed on Mac OS X systems using MacPorts[c].

Once the autotools are available (as well as other required software such as Cygwin or MacPorts), the user can proceed with the installation of Thot by following the next sequence of steps:

1. Obtain the package using git:

```
$ git clone https://github.com/daormar/thot.git
```

   Additionally, Thot can be downloaded in a zip file[d].

2. `cd` to the directory containing the package's source code and type `./reconf`.
3. Type `./configure` to configure the package.
4. Type `make` to compile the package.
5. Type `make install` to install the programs and any data files and documentation.
6. You can remove the program binaries and object files from the source code directory by typing `make clean`.

By default the files are installed under the `/usr/local` directory (or similar, depending of the OS you use); however, since Step 5 requires root privileges, another directory can be specified during Step 3 by typing:

```
$ configure --prefix=<absolute-installation-path>
```

For example, if `user1` wants to install the Thot package in the directory `/home/user1/thot`, the sequence of commands to execute should be the following:

[a] https://github.com/daormar/thot/
[b] https://www.cygwin.com/
[c] https://www.macports.org/
[d] https://github.com/daormar/thot/archive/master.zip

```
$ ./reconf
$ configure --prefix=/home/user1/thot
$ make
$ make install
```

The installation process also creates three directories with additional information:

- **${PREFIX}/share/thot/cfg_templates**: contains configuration files to be used with different Thot utilities (see Chapter 3 for more details).
- **${PREFIX}/share/thot/doc**: contains the documentation of Thot, which currently consists in the Thot manual (`thot_manual.pdf`).
- **${PREFIX}/share/thot/toy_corpus**: contains a very small parallel corpus to make software tests. The directory includes both raw and preprocessed versions of the corpus (see Sections 2.2 and 3.2 for more details). This corpus may also be useful for new Thot users trying to get familiar with the toolkit functionality.

See the `INSTALL` file in the directory where Thot has been downloaded for more information.

1.2 Checking Package Installation

Once the package has been installed, it is possible to perform basic checkings in an automatic manner so as to detect portability errors. For this purpose, the following command can be executed:

```
$ make installcheck
```

2 Preliminary Notes

2.1 Corpus Partition

SMT systems use parallel corpora to train and tune the model parameters. After the parameters have been estimated, the resulting models are used to obtain the target translations. The completion of these tasks require the generation of a corpus partition composed of three different sets:

- **Training set**: the training set is used to train the different statistical models involved in the translation. It is typically composed by many thousands or even millions of sentence pairs (greater training set sizes usually allow us to increase translation quality).
- **Development set**: the development set is used for parameter tuning (it is not used in the initial training stage). This set is typically composed of a few thousand sentence pairs (1 000 or 2 000 are usual in common translation tasks).
- **Test set**: the test set is used to compute automatic evaluation measures using the target sentence as reference sentences. This set is often composed of a few thousand sentence pairs in the same way as the development set.

In Thot it is assumed that, for a specific set, there will be two parallel files, one related to the source language and another related to the target sentence.

2.2 File Naming Conventions

To simplify the usage of some translation tools offered by Thot, it is useful to define a specific naming convention for the files containing the partition of the parallel corpus. In particular, the names will be composed by a prefix specific to the source or the target languages, and a suffix identifying whether the file belongs to the training, development or test set.

To illustrate this file naming convention, we can look at the files that compose the Spanish to English toy corpus included in the package (installed under the `${PREFIX}/share/thot/toy_corpus` directory, see Section 1.1):

- {**sp**}|{**en**}**.train**: the files `sp.train` and `en.train` compose the training set of the toy corpus, which is available in the Spanish and English languages.
- {**sp**}|{**en**}**.dev**: `sp.dev` and `en.dev` correspond to the development set.
- {**sp**}|{**en**}**.test**: finally, `sp.test` and `en.test` correspond to the test set.

3 Quick Usage Examples

3.1 Corpus Downloading

To carry out a simple translation experiment, we are going to execute an Spanish to English translation task focused on the tourist domain.

For this purpose, we create a new directory called `thot`, and change the directory to it:

```
mkdir thot
cd thot
```

After that, we download a compressed file with the data :

```
wget http://daormar.github.io/thot/quick_guide/data.tar.gz
```

Once we have downloaded the corpus, we decompress the file and inspect the new directory that is created:

```
tar -zxvf data.tar.gz
ls data
```

3.2 Corpus Preprocessing

In common translation tasks, it is often interesting to preprocess the available parallel texts to make the translation process easier.

3.2.1 Tokenization

Thot provides the `thot_tokenize` tool, which can be used to tokenize texts. The following list of commands obtain a tokenized version of the corpus files:

```
thot_tokenize -f data/sp.train > data/sp_tok.train
thot_tokenize -f data/sp.dev > data/sp_tok.dev
thot_tokenize -f data/sp.test > data/sp_tok.test

thot_tokenize -f data/en.train > data/en_tok.train
thot_tokenize -f data/en.dev > data/en_tok.dev
thot_tokenize -f data/en.test > data/en_tok.test
```

3.2.2 Lowercasing

After tokenizing the corpus, it can also be useful to lowercase it. This frees the translation system from the task of finding an appropriate capitalization for the translated text.

To lowercase text, Thot provides the `thot_lowercase` tool. The following commands lowercase the tokenized source files:

```
thot_lowercase -f data/sp_tok.train > data/sp_tok_lc.train
thot_lowercase -f data/sp_tok.dev > data/sp_tok_lc.dev
thot_lowercase -f data/sp_tok.test > data/sp_tok_lc.test

thot_lowercase -f data/en_tok.train > data/en_tok_lc.train
```

```
thot_lowercase -f data/en_tok.dev > data/en_tok_lc.dev
thot_lowercase -f data/en_tok.test > data/en_tok_lc.test
```

3.3 Training and Tuning

3.3.1 Language Model Training

Thot provides the `thot_lm_train` tool, which can be used to train language models.

The following command line trains a 3-gram language model for the English tokenized and lowercased training set of the corpus, using the `-unk` option, storing the results in the `lm` directory:

```
train_corpus=data/en_tok_lc.train
thot_lm_train -c ${train_corpus} -o lm -n 3 -unk
```

The command also generates a model descriptor in the output directory name `lm_desc` that will be useful to generate configuration files.

3.3.2 Translation Model Training

Thot incorporates the `thot_tm_train` tool, useful to train phrase models.

The following command line trains a phrase model for the tokenized and lowercased training set of the corpus, storing the results in the `tm` directory:

```
src_train_corpus=data/sp_tok_lc.train
trg_train_corpus=data/en_tok_lc.train
thot_tm_train -s ${src_train_corpus} -t ${trg_train_corpus} -o tm
```

The command also generates a model descriptor in the output directory name `tm_desc` that can be used to generate configuration files.

3.3.3 Basic Configuration File Generation

Thot uses configuration files to simplify the access to the toolkit functionality. These configuration files provides information about the parameters used by the Thot translation system, including the location of the translation and language models, the set of log-linear model weights, parameters relevant to the way in which the search process is carried out, etc.

Configuration files can also be generated by means of the `thot_gen_cfg_file` command. For this purpose, it is necessary to provide the descriptors of both the language and translation models.

The following command line generates a basic Thot configuration file and writes it to the file `before_tuning.cfg`:

```
thot_gen_cfg_file lm/lm_desc tm/tm_desc > before_tuning.cfg
```

3.3.4 Parameter Tuning

After training the language and translation models, it is necessary to execute a parameter tuning stage. For this purpose, a development corpus separated from the training corpus is required.

The `thot_smt_tune` tool allows to perform parameter tuning. For this purpose, it is necessary a Thot configuration file and a development corpus.

The following command line tunes the system given in the `before_tuning.cfg` file, for the tokenized and lowercased development set of the corpus, storing the results in the `smt_tune` directory:

```
src_dev_corpus=data/sp_tok_lc.dev
trg_dev_corpus=data/en_tok_lc.dev
thot_smt_tune -c before_tuning.cfg -s ${src_dev_corpus} -t ${trg_dev_corpus} \
              -o smt_tune
```

After the successful execution of `thot_smt_tune`, the configuration file corresponding to the tuned system will be stored in `smt_tune/tuned_for_dev.cfg`.

3.3.5 Phrase Model Filtering

Phrase models are composed of millions of parameters when they are estimated from large training corpora, making impossible to store them in main memory when using regular computer hardware. One simple solution to this problem when the set of sentences to be translated is known beforehand is to filter those phrase model parameters that are relevant to carry out the translation process.

The `thot_prepare_sys_for_test` tool allows to filter the parameters of a phrase model. For this purpose, it is necessary a Thot configuration file and a file with the sentences to be translated.

The following command line filter the phrase model given in the `smt_tune/tuned_for_dev.cfg` file (obtained after the parameter tuning step) for the tokenized and lowercased test set of the corpus, storing the results in the `systest` directory, where the `test_specific.cfg` file will be generated:

```
src_test_corpus=data/sp_tok_lc.test
thot_prepare_sys_for_test -c smt_tune/tuned_for_dev.cfg \
                          -t ${src_test_corpus} -o systest
```

3.4 Generating Translations

Thot incorporates tools to translate a set of sentences in a fully-automatic way. This task can be carried out using both, command line and client-server tools.

3.4.1 Basic Tools

The `thot_decoder` tool allows to generate translations for a given test set. For this purpose, a Thot configuration file and a file with the sentences to be translated should be provided.

The following command uses the system configuration provided in the `systest/test_specific.cfg` file to translate the tokenized and lowercased test set of the corpus, storing the results in the `output` file:

```
src_test_corpus=data/sp_tok_lc.test
thot_decoder -c systest/test_specific.cfg -t ${src_test_corpus} -o output
```

3.4.2 Client-Server Tools

The translation functionality mentioned above is also included in the client-server architecture provided by Thot. This includes two basic tools: `thot_server` and `thot_client`.

The `thot_server` tool implements a fully-fledged SMT system. The most relevant input parameter that has to provided, the `-c` parameter, is the name of the configuration file.

On the other hand, `thot_client` can be used to request translations to the server. `thot_client` requires the IP address where the server is being executed using the `-i` option, as well as the sentence to be translated, that is provided by means of the `-t` option.

The following example shows how to obtain the translation of a test sentence of the corpus using the client-server architecture for the `systest/test_specific.cfg` configuration file:

```
thot_server -c systest/test_specific.cfg & # start server
thot_client -i 127.0.0.1 -t 'voy a marcharme hoy por la tarde .'
```

3.5 Output Postprocessing

Once the output of the system have been obtained, it may be necessary to recase and/or detokenize the text.

3.5.1 Recasing

If the initial corpus was lowercased, it will be necessary to recase the output of the system. The Thot toolkit incorporates its own recasing tool: `thot_recase`.

One possible option to supply the raw text file would be simply passing the target training set. However, a better option could be to also include the target development set and the source test set as well (the model can take advantage of information in the source language for specific words). If the user adheres to the file naming conventions described in Section 2.2, then it is very easy to generate this raw text file by means of the `thot_gen_rtfile` tool, which requires the prefix of the source and target corpus files:

The following commands allows us to recase the output given by the decoder in the example shown in Section 3.4 (since the corpus was tokenized during the corpus preprocessing stage, we should provide the prefix corresponding to the tokenized files):

```
thot_gen_rtfile -s data/sp_tok -t data/en_tok > rtfile_rec
thot_recase -f output -r rtfile_rec > output_rec
```

3.5.2 Detokenization

The toolkit also incorporates a tool to detokenize texts: `thot_detokenize`.

Again, the raw text file required by `thot_detokenize` can be generated by means of the `thot_gen_rtfile` tool, as it was explained above for the recaser.

The following commands allows us to detokenize the recased text obtained in the example of Section 3.4:

```
thot_gen_rtfile -s data/sp -t data/en > rtfile_detok
thot_detokenize -f output_rec -r rtfile_detok > output_rec_detok
```

3.6 Output Evaluation

After translating a test set, the translation quality can be evaluated using automatic measures provided that there exist reference translations for each source sentence. Thot implements some tools for this purpose:

- **`thot_calc_bleu`**: obtains the BLEU (bilingual evaluation understudy) measure.

- **thot_calc_wer**: calculates the WER (word error rate) measure (the number of substitutions, insertions and deletions that are required to convert the system translation into the reference sentence).

The following command allows to evaluate the translation quality of the translations obtained in Section 3.4:

```
trg_test_corpus=data/en_tok_lc.test
thot_calc_bleu -r ${trg_test_corpus} -t output
```

3.7 Advanced Functionality

This section explains how to access advanced features included in the Thot toolkit.

3.7.1 Interactive Machine Translation

Thot implements interactive machine translation functionality, which allows to generate suffixes that complete the prefixes validated by the user.

The interactive machine translation functionality implemented in Thot can be accessed by means of the `thot_server` and `thot_client` tools. This functionality includes obtaining the initial translation of a source sentence and obtaining the suffix that best completes a prefix given by the user.

Before using the interactive machine translation functionality, it is necessary to initialize the server. For this purpose, the `thot_server` tool requires the name of configuration file that is supplied by means of the `-c` parameter.

On the other hand, requests to the server can be sent by means of `thot_client`. As basic parameter, `thot_client` requires the IP address where the server is being executed using the `-i` option. Thot maintains state information through the different interactions between the user and the system. Some of the information that is mantained is specific to the user and hence it is important to use different user identifiers for different interactive translation sessions. The user identifier can be provided by means of the `-uid` option. To interactively translate a sentence, the `-sc` option is used to start the translation process. After that, new strings can be added to the previously existing prefix by means of the `-ap` option. It is important to point out that the server works at character level. Finally, the `-rp` option can be used to reset the prefix.

The following example shows how to access the basic interactive machine translation functionality provided by Thot using the client-server architecture:

```
thot_server -c systest/test_specific.cfg & # start server
thot_client -i 127.0.0.1 -uid 0 -sc 'me marcho hoy por la tarde .'
thot_client -i 127.0.0.1 -uid 0 -ap 'I am '
```

```
thot_client -i 127.0.0.1 -uid 0 -ap 'le'
# NOTE: at this point, the user prefix provided to the system is 'I am le'
thot_client -i 127.0.0.1 -uid 0 -rp # reset prefix
```

3.7.2 Online Learning

Thot incorporates techniques that allow to incrementally update the parameters associated to the features of a state-of-the-art log-linear model.

The online learning functionality implemented in Thot can be accessed by means of the of the `thot_server` and `thot_client` tools. This functionality includes processing a single training pair and printing the models.

Prior to use the online learning functionality, the server should be initialized. For this purpose, the `thot_server` tool requires the name of configuration file that is supplied by means of the `-c` parameter.

On the other hand, requests to the server can be sent by means of `thot_client`. As basic parameter, `thot_client` requires the IP address where the server is being executed using the `-i` option. In addition to this, the `-tr` option can be used to specify the training pair to be processed and the `-pr` option allows to print the updated models to files. Printing the models causes the previous ones to be overwritten.

The following example shows how to process a new training pair and print the models using the client-server architecture:

```
thot_server -c systest/test_specific.cfg & # start server
thot_client -i 127.0.0.1 -tr 'esto es una prueba' 'this is a test'
thot_client -i 127.0.0.1 -pr # print models (previous ones are overwritten)
```