Report: Dynamic Parameters for Moses

Marcin Junczys-Dowmunt

September 27, 2011

Contents

Chapter 1

General Assumptions

1.1 Introduction

This report describes the progress of work on the extension of the open-source machine translation system Moses with the possibility to set sentence-related parameters dynamically. The final goal lies in a better integration with the TAPTA application developed at the Global Database Service at WIPO. Dynamic parameters are meant to allow the modification of Moses options and translation model weights during runtime on a sentence-by-sentence basis. TAPTA allows the user to work with the translation of a single sentence on the level of the whole sentence and on subsentence level. With a special interface the user will be able to tweak the results of the underlying machine translation engine directly. Dynamic parameters will allow to set among others the verbosity for a single sentence, search-related parameters, weights for language models, word-penalties etc. This can be done for a single sentence with a syntax similar to the following:

```
<specOpt weights="tm:1:2 0.4 lm 0.2" switches="-s 20 -v 2"/> #t energy
optimized thermodynamic cycle #t
```

where the sentence `#t energy optimized thermodynamic cycle #t` is translated by Moses using modified standard settings. In this example, the stack size is decreased to 20, verbosity level is set to 2, and translation model weights are partially altered. Weights and parameters that are not specified are inherited from the basic configuration and left unchanged. The changed parameters affect only the given sentence. Another sentence has to supply its own parameters or it will be translated with standard settings.

Parameters can be obtained in two ways: by a human try-and-error procedure or by automatic optimization. We assume that both approaches can useful for the WIPO machine translation system.

In this document we will justify the taken approach and give a short reference how to use the newly implemented features. We will concentrate only on the Moses-part of the implementation. The necessary changes to TAPTA are not in the scope of this report.

1.2 Dynamic Parameters vs Translation Systems

Moses offers a functionality to configure several translation systems with separate language models, phrase tables etc. and corresponding feature weights. It could be argued that this mechanism can be successfully used instead of dynamic parameters. This is true for scenarios in which we know in advance what parameters will be used. Then we can fix different sets of parameters and switch between the sets, achieving a similar effect to dynamic parametrization. By default however, Moses does not have a mechanism for assigning command line options to translation systems. Only feature functions and feature weights can be assigned.

This approach has however other serious drawbacks: in the implementation of Moses all weights are stored in a single vector. This vector is used for the calculation of hypothesis scores and the longer this vector the more time the calculations take, since methods from linear algebra are performed on the whole length of that vector. With tens of possible translation systems this vector will increase in size accordingly. Another aspect is memory consumption, since Moses does not share phrase tables, generation tables and reordering models between translation systems. All of them have to be present in memory all the time. This can result in serious overhead even if disk-based implementations are used.

Our implementation of dynamic parameters does not suffer from these problems. For each sentence a shallow copy of all static data in the application is created which in the context of the given sentence shadows the static data of the decoder. During our experiments, we did not notice any performance drops due to that copying procedure. The weight vector does not grow due to that action. It can be easily modified in-place without affecting the original weight vector. With dynamic parameters an unlimited number of translation systems can be simulated as long as differences between the systems are restricted to translation model weights and sentence-related decoding options.

Another argument for dynamic parameters is, that translation models cannot be used if parameters are not known in advance, since translation systems need to be defined at start-up. Dynamic parameters are meant to be used during runtime and can take any meaningful value. This allows for the continuous employment of a production environment as it is the case here at WIPO. Even if new results are available due to later experiments they can be used immediately with an existing and running system.

1.3 Thread Safety

An important aspect with dynamic parameters is thread safety. We need to make sure that setting a parameter dynamically for one sentence in a certain thread will not affect the translation process of another sentence in another thread. This is achieved by assigning a shallow copy of the static data present in Moses to a single thread. Only this thread has access to the copy and modifications carried out on the cloned static data is not reflected in other threads. A single sentence is always translated in one thread. Once the translation has been finished the thread is again assigned the original static data object. The shallow copy is destroyed together with the previous modifications.

1.4 Example Scenario

In this section we will shortly discuss a very basic experiment. We investigate the effects of optimizing translation model parameters for separate classes of sentences. This is an example for the automatic obtainment of parameters.[1] For the first experiment we divide the development data and the unseen test data into four categories `#c, #d, #s, #t` (configuration *cats*).

[1]In contrast to manually chosen parameters.

Configuration	bleu
$baseline \times lm_1$	48.01
$cats \times lm_1$	48.13
$cats \times lm_1 lm_2 lm_3$	48.61
$length \times lm_1$	48.34
$length \times lm_1 lm_2 lm_3$	48.49

Table 1.1: enfr.0-0.n100

Here #t stands for titles, #d for the first sentence of a description, #c for a sentence part, #s for all remaining sentences. We test this configuration using a general language model lm_1 built from the complete data (without development and test set) and with two additional language models built from titles (lm_2) and abstracts (lm_3). For the second experiment we divide the data by sentence length (configuration *lengths*) and pair it again with different language models. All configurations and classes are optimized separately with MERT. We use the most recent single-factored English-French translation model for our experiments.

Table 1.1 summarizes some of the obtained results. It seems that data separation leads to improved results.[2] How does this relate to dynamic parametrization? The prior division into classes can be executed by an external program (e.g. a simple Perl script) by adding parameters for each sentence according to its class membership. This way a single test set does not need to be divided into parts and can be translated by the same Moses process. Results from different runs do not require the running system to be stopped and renewed as it would be the case with the aforementioned static translation systems.

1.5 Future Work

We plan to extend our modifications to Moses with the possibility to use relative values for switches and weights, i.e. of the form `-s +20%` or `-s +20`. Using this notion the current value of a parameter will by increased or decreased, but not set absolutely.

As for the experimental part of future work it is necessary to continue and extend the preliminary results described in the short experiment above. We will investigate different settings and classes and try to verify the improvement achieved by optimizing parameters for separate classes. The experiments will be carried out for other translation directions and language pairs. That way we hope to show the general usefulness of the implemented dynamic parameters.

[2]These results are preliminary and need to be confirmed for a greater number of repetitions of the MERT training. MERT is a stochastic optimizer that can reach strongly differing results between two runs. Now we need to confirm that the effects are not due to pure chance, the differences lie within the variance for MERT results. Furthermore we need to investigate the performance of a second baseline configured as $baseline \times lm_1 lm_2 lm_3$ in order to provide a fair comparison.

Chapter 2

Short Reference

In the following sections the new dynamic parameter features added to Moses will be covered in some more detail.

2.1 Synopsis

```
<specOpt/> #t energy optimized thermodynamic cycle #t
<specOpt system="test1"/> #t energy optimized thermodynamic cycle #t
#t energy optimized <specOpt system="test1"/> thermodynamic cycle #t
<specOpt weights="tm:1:2 0.4 lm 0.2"/> #t energy optimized thermodynamic
cycle #t
<specOpt name="predefined parameters"/> #t energy optimized thermodynamic
cycle #t
<specOpt switches="-s 20 -v 2"/> #t energy optimized thermodynamic cycle #t
<specOpt system="test1" weights="tm:1:2 0.4 lm 0.2" switches="-s 20 -v 2"/>
#t energy optimized thermodynamic cycle #t
```

The tag `specOpt` is required to be an empty, correct XML-tag. It can be placed anywhere within the sentence and will affect the way the whole sentence is translated regardless of the position of the tag.

There are three parameters: `system, switches, weights, name`. The `system` parameter allows to set explicitly which translation system is to be used as a basis. The explicitly named system will then be modified and used for translation. As stated in the first chapter, the modifications are not permanent and fall back to their prior setting once the sentence has been translated. Modifications are also thread-safe, a sentence that uses the same system will always be translated with the correct settings for the system and the sentence, regardless of the modifications performed for another sentence. If no system name is given this is equivalent to setting `system="default"`.

The remaining two parameters will be discussed in the next two sections.

2.2 Setting Application Switches

Switches can be used in the same way as normal command-line switches. When a new command-line switch is added to Moses it becomes also available to use with the described extension. However, not all switches have an observable effect. In general only switches that affect the way how a single sentence is translated can be used meaningfully. It is for instance not possible to change the location of the n-best list file or to read in a new phrase table or language model. Meaningful parameters are, for instance, the level of verbosity, stack size, various thresholds, or the choice of the used decoding algorithm. It is also possible to set translation model weights through switches, but as in the case of the real command-line it is necessary to specify all required weights at one time. Single weights cannot be set if the particular feature function requires more than one weight, e.g. the `-tm` switch requires at least 5 weights. Similarly, if two phrase tables are used, it is necessary to give all weights for all phrase tables, in that case `-tm` requires 10 weights.

Example

```
<specOpt switches="-s 20 -v 2 -w 0.4 -tm 0.34 0.31 0.12 -0.43 0.54 -mp"/> #t
energy optimized thermodynamic cycle #t
```

2.2.1 Supported command-line switches

What follows is an selection of supported command line switches. The descriptions have been taken from the original Moses documentation.

```
-beam-threshold (b): threshold for threshold pruning
-clean-lm-cache: clean language model caches after N translations (default N=1)
-consensus-decoding (con): use consensus decoding (De Nero et. al. 2009)
-cube-pruning-diversity (cbd): How many hypotheses should be created for each
    coverage. (default = 0)
-cube-pruning-pop-limit (cbp): How many hypotheses should be popped for each
    stack. (default = 1000)
-disable-discarding (dd): disable hypothesis discarding
-distortion-limit (dl): distortion (reordering) limit in maximum number of
    words (0 = monotone, -1 = unlimited)
-drop-unknown (du): drop unknown words instead of copying them
-early-discarding-threshold (edt): threshold for constructing hypotheses
    based on estimate cost
-max-chart-span: maximum num. of source word chart rules can consume
    (default 10)
-max-phrase-length: maximum phrase length (default 20)
-max-trans-opt-per-coverage: maximum number of translation options per input
    span (after applying mapping steps)
-mbr-scale: scaling factor to convert log linear score probability in MBR
    decoding (default 1.0)
-mbr-size: number of translation candidates considered in MBR decoding
    (default 200)
-minimum-bayes-risk (mbr): use miminum Bayes risk to determine best translation
-monotone-at-punctuation (mp): do not reorder over punctuation
-persistent-cache-size: maximum size of cache for translation options (default
    10,000 input phrases)
-phrase-drop-allowed (da): if present, allow dropping of source words
```

```
-print-alignment-info: Output word-to-word alignment into the log file.
    Word-to-word alignments are taken from the phrase table if any. Default
    is false
-print-all-derivations: to print all derivations in search graph
-report-all-factors: report all factors in output, not just first
-report-all-factors-in-n-best: Report all factors in n-best-lists. Default
    is false
-report-segmentation (t): report phrase segmentation in the output
-search-algorithm: Which search algorithm to use. 0=normal stack, 1=cube
    pruning, 2=cube growing. (default = 0)
-source-label-overlap: What happens if a span already has a label. 0=add more.
    1=replace. 2=discard. Default is 0
-stack (s): maximum stack size for histogram pruning
-stack-diversity (sd): minimum number of hypothesis of each coverage in
    stack (default 0)
-translation-option-threshold (tot): threshold for translation options
    relative to best for input phrase
-ttable-limit (ttl): maximum number of translation table entries per
    input phrase
-use-persistent-cache: cache translation options across sentences
    (default true)
-verbose (v): verbosity level of the logging
-weight-d (d): weight(s) for distortion (reordering components)
-weight-e (e): weight for word deletion
-weight-generation (g): weight(s) for generation components
-weight-i (I): weight(s) for word insertion - used for parameters from
    confusion network and lattice input links
-weight-l (lm): weight(s) for language models
-weight-lex (lex): weight for global lexical model
-weight-t (tm): weights for translation model components
-weight-u (u): weight for unknown word penalty
-weight-w (w): weight for word penalty
```

2.3 Setting Translation Model Weights

It was mentioned in the last section that it not possible to set single translation model weights with the help of command-line switches if a feature function requires more than one weight. The `weights` parameter can be to overcome that. Any weight of any feature function can be addressed directly and set separately.

2.3.1 General format

The value of the `weights` parameter consists of a space delimited sequence of references to weights and their corresponding values. The general format is as follows:

```
-<short-weight-name>:<feature-index>:<weight-index> <float>
```

Examples

```
-tm:0:0 0.34      # Set 1st weight of first phrase table to 0.34
-tm:1:4 0.45      # Set 5th weight of first phrase table to 0.45

-lm:1:0 -0.12     # Set weight of second language model to -0.12

-w:0:0  -1.2      # Set word penalty to -1.2
```

Any number of weights in any order can be set that way. Repeated values will overwrite the previous occurrences. If a reference addresses a weight that does not exist the value is simply ignored.

Example:

```
<specOpt weights="-tm:0:0 0.34 -tm:0:1 0.45 -lm:0:0 0.3 -tm:1:3 0.32 -w:0:0 -1.2"/>
#t energy optimized thermodynamic cycle #t
```

2.3.2 Short format

In order to make referencing easier, there are default values for `<featureindex>` and `<weight-index>` and both can be omitted in certain circumstances.

Examples

```
-tm 0.34          # The same as -tm:0:0 0.34

-tm:1 0.34        # The same as -tm:1:0 0.34

-lm -0.12         # The same as -lm:0:0 -0.12

-w -1.2           # The same as -w:0:0 -1.2
```

However, if there are several features of the same kind, you have to use the full format if you want to set weights for any feature with an index greater than 0.

```
-lm:1:2 -0.12
```

It also possible to provide more than one weight per switch. If the number of weights given exceeds the number of scores for the chosen feature than the weights are passed on the the next feature function of the same type.

For instance if we have a distance based reordering model and a lexical reordering model we can set the weights for both of them by providing

```
-d 0.3 0.3 0.3 0.3 0.3 0.3 0.3
```

which is then equivalent to the following:

```
-d 0.3 -d:1 0.3 0.3 0.3 0.3 0.3 0.3
```

or more verbose:

```
-d:0:0 0.3 -d:1:0 0.3 -d:1:1 0.3 -d:1:2 0.3 -d:1:3 0.3 -d:1:4 0.3 -d:1:5 0.3 -d:1:6 0.3
```

Example

The same line as above can now be written:

```
<specOpt weights="-tm 0.34 0.45 -lm 0.3 -tm:1:3 0.32 -w -1.2"/> #t energy
optimized thermodynamic cycle #t
```

2.3.3 Weight names

The short names of the weights are the same as in the n-best lists.

```
d       # distortion model (generally 1 or 6 weights)
w       # word penalty (1 weight)
tm      # translation model (5 weights)
lm      # language model (1 weight)
g       # genertion model (2 weights)
```

2.4 Named settings

It is possible to provide a name for a given setting. If this name is called again without any other parameters the parameters given before will be reused. If a name is given with parameters the old settings will be discarded and replaced.

```
# define name
<specOpt name="some params" weights="-tm 0.34 0.45 -lm 0.3 -tm:1:3 0.32
-w -1.2"/> #t energy optimized thermodynamic cycle #t

# call by name
<specOpt name="some params"/> #t energy optimized thermodynamic cycle #t

# overwrite name
<specOpt name="some params" weights="-w -1.2"/> #t energy
optimized thermodynamic cycle #t
```

It is also possible to predefine parameter settings in an external file specifying one `<specOpt .../>` tag per line. The decoder will read in the file with the following command line option:

```
moses -f moses.ini -wipo-specopt-file specopts.txt
```