PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 96 OCTOBER 2011 49–58

Ncode: an Open Source Bilingual N-gram SMT Toolkit

Josep M. Crego[a], François Yvon[ab], José B. Mariño[c]

[a] LIMSI-CNRS, BP 133, 91430 Orsay Cedex, France
[b] Université Paris-Sud, 91430 Orsay, France
[c] Universitat Politècnica de Catalunya, Jordi Girona S/N, Barcelona 08034, Spain

Abstract

This paper describes Ncode, an open source statistical machine translation (SMT) toolkit for translation models estimated as n-gram language models of bilingual units (*tuples*). This toolkit includes tools for extracting tuples, estimating models and performing translation. It can be easily coupled to several other open source toolkits to yield a complete SMT pipeline. In this article, we review the main features of the toolkit and explain how to build a translation engine with Ncode. We also report a short comparison with the widely known Moses system. Results show that Ncode outperforms Moses in terms of memory requirements and translation speed. Ncode also achieves slightly higher accuracy results.

1. Introduction

This paper describes Ncode, an open source statistical machine translation decoder and its companion tools. Ncode implements the bilingual n-gram approach to SMT as described in (Mariño et al., 2006; Crego and Mariño, 2007), which can be seen as an alternative to the standard phrase-based approach (Zens et al., 2002). Ncode main features include the use of multiple n-gram language models estimated over bilingual units, source words and/or target words or any factor decomposition, lexicalized reordering, several tuple (unigram) models, *etc.*. As for nearly all current statistical approaches to machine translation, these models are embedded in a linear model combination. Ncode splits the reordering and decoding problems of SMT in two separate modules, aiming at better tackling each of the problems. However, hard reordering decisions are avoided by means of using permutation lattices.

Corresponding author: jmcrego@limsi.fr
Cite as: Josep M. Crego, François Yvon, José B. Mariño. Ncode: an Open Source Bilingual N-gram SMT Toolkit. The Prague Bulletin of Mathematical Linguistics No. 96, 2011, pp. 49–58. doi: 10.2478/v10108-011-0010-5.

The toolkit implements algorithms for tuple extraction, modeling estimation and translation. Several algorithms for optimization and evaluation are borrowed from distinct open source projects, embedded in our toolkit to accurately work with our translation engine. The decoder takes advantage of multi-threaded architectures, which are quickly becoming the norm. As far as memory is concerned, major improvements have been obtained by replacing the SriLM[1] interface (used in previous versions) with the new and much leaner kenlm[2] libraries. The toolkit is mainly written in *C++* and *Perl*, with special attention to clean code, extensibility and efficiency, and is available under an open-source license. It is mainly developed to run on Linux systems. Prerequisites to compile the decoder are kenlm and OpenFst[3] libraries. Prerequisites to run the entire system are SriLM and the minimum error rate training (Och and Ney, 2002) implementation available in the Moses[4] SMT toolkit. Note that the toolkit is able to build translation systems starting from a parallel set of word-aligned sentences and typically employs part-of-speeches to learn rewrite rules. Therefore, although they are not directly required by our SMT system, a word alignment and symmetrization algorithms as well as POS taggers for the source and target languages are needed to perform the entire SMT pipeline.

The toolkit was originally developed at UPC, further extended at LIMSI to its current state. It has been successfully used in a number of machine translation evaluations. A detailed description of the system with examples and full documentation is available in the LIMSI's web site[5]. The rest of this paper is organized as follows. In Section 2, we briefly introduce the bilingual n-gram approach to statistical machine translation. In Sections 3, 4 and 5, we detail the main components of the toolkit: training, decoding and tuning. After a short comparison with Moses in Section 6, we finally draw conclusions in Section 7.

2. Bilingual N-gram Approach to Statistical MT

The bilingual n-gram approach to SMT has been derived from the finite-state perspective (Casacuberta and Vidal., 2004). However, while translation models are implemented as weighted finite-state transducers in the finite-state perspective, our approach implements translation models as simple n-gram language models. The elementary translation units are *tuples*, that is pairs of variable-length sequences of source and target words. Hence, the translation model defines probability over sequences of tuples. Training such a translation model requires that (i) the source and

[1] http://www.speech.sri.com/projects/srilm/

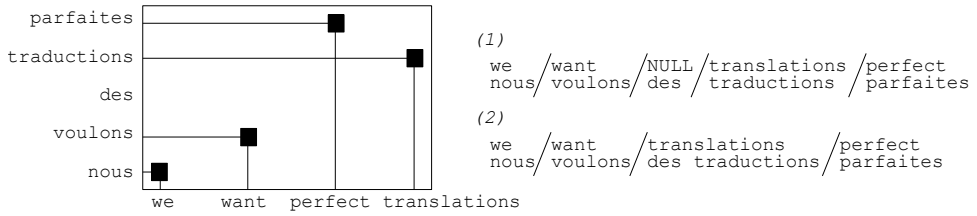
[2] http://kheafield.com/code/kenlm/

[3] http://www.openfst.org

[4] http://www.statmt.org/moses/

[5] http://ncode.limsi.fr

Figure 1. Tuple extraction from a word-aligned sentence pair (English-French).



target side tuples are synchronized, *i.e.* that they occur in the same order in their respective languages and (ii) a joint segmentation of the source and target sentences in tuples is available, which uncovers the tuple boundaries. Given a parallel training corpus, two pre-processing steps are thus necessary to meet these requirements. First, word alignments are derived for each sentence pair; based on this information, a joint segmentation of the source and target sentences in tuples is then produced. The segmentation in tuples is made (almost) deterministic by enforcing the following constraints: (i) no word inside a tuple can be aligned to a word outside the tuple; (ii) segmentation must respect the order of words as they occur on the target side. Reordering is permitted in the source side so as to synchronize the source and target sides of a sentence; and (iii) no smaller tuples can be found without violating the previous constraints. Figure 1 presents a simple example illustrating the unique tuple segmentation for a given word-aligned pair of sentences. Note that the English source words *perfect* and *translations* have been reordered in the final tuple segmentation, while the French target words are kept in their original order. The resulting sequence of tuples *(1)* is further refined *(2)* to avoid *NULL* words in the source side of tuples. Refer to (Crego and Mariño, 2007) for further details on the tuple extraction process.

The bilingual n-gram language model expects synchronized tuple streams in training; likewise, it produces synchronized streams in inference. This means that the input source stream has to be reordered *prior to translation*, so as to reproduce the word order changes introduced during the training process. In our system, several possible reorderings of the source are considered in parallel. To this end, the sentence to be translated is first turned into a word lattice containing a set of promising reordering hypotheses. It is then pretty straightforward to search this lattice in a monotonous fashion for the best translation hypothesis. See (Crego and Mariño, 2007) for further details on word reordering.

3. Training a NCODE SMT system

In this section, we outline the training process of a NCODE system. Training basically involves the estimation of the set of models used by the decoder to perform machine translation. The script `training.perl` implements the training process, assuming source and target files, as well as the corresponding word alignment. It performs the following steps:

Tuple extraction From a word-aligned training bitext, tuples are extracted following the algorithm sketched in Section 2.

Tuple refinement Tuples with a *NULL* source side are then discarded by merging the unaligned target word with either the previous or the next unit (see Figure 1).

Tuple pruning and uncontextualized scores As word alignments are often noisy, we then filter tuples using a set of simple constraints. A tuple is discarded if it exceeds a maximum number of source (or target) words (`--max-tuple-length`); or a maximum ratio of source/target length (`--max-tuple-fert`). Additionally, only the n best translation choices for each tuple source side are considered (`--tuple-nbest`). Four scores are then associated with each tuple, corresponding to conditional probabilities computed as relative frequencies:

$$P_{rf}(e,f) = \frac{count(f,e)}{\sum_{f'} count(f',e)} \quad ; \quad P_{rf}(f,e) = \frac{count(f,e)}{\sum_{e'} count(f,e')} \tag{1}$$

where f and e respectively denote the tuple source and target side.
Word-based lexicon weights are also computed for each translation unit:

$$P_{lw}(e,f) = \frac{1}{(J+1)^I} \prod_{i=1}^{I} \sum_{j=0}^{J} P_{lex}(e,f) \quad ; \quad P_{lw}(f,e) = \frac{1}{(I+1)^J} \prod_{j=1}^{J} \sum_{i=0}^{I} P_{lex}(f,e) \tag{2}$$

where the probability distribution P_{lex} is computed based on counts using the word alignments.

Bilingual n-gram LM The training bitext expressed in the form of tuples is used to estimate a standard n-gram language model. It is estimated using SRILM, any of its modeling available options can be used (use `--options-bm`). When several bitexts are available, several LMs can be learned and used in parallel. In this case, `--name-src-bm` and `--name-trg-bm` are used to identify the potentially multiple language model files.
Tuples are typically built from words in the source and target sides, however, different factors may be used. *i.e.* tuples may be built from source POS tags and target lemmas (use `--train-src-bm train.f.pos --train-trg-bm train.e.lem`). Source and target factored training files must match with the alignment file, and are bound to contain the same number of sentences and words per sentence. Notice also that our current implementation considers **one single** factored tuple associated with each original word-based tuple (enabling a straightforward

implementation of factored language models), even though several different instances may exist. In such case, the most frequent is only considered.

Reordering rules Rewrite rules are automatically learned from the bitext word alignments. Following on the example of Figure 1, the rule *perfect translations* $\rightsquigarrow$ *translations perfect* produces the swap of the English words that is observed for the French and English pair. Note that such simple reordering patterns can be modeled using finite-state transducers. Typically, POS tags are used to increase the generalization power of such rules (use `--train-src-rules train.f.pos`), however, any other factor form can be used. Again, the source file used to extract reordering rules must be parallel to the original source training file.
In order to discard rules derived from noisy alignments, rules are pruned according to a length (`--max-rule-length`) and minimum probability threshold (`--max-rule-cost f`). The probability of each rule is estimated as:

$$P(f \rightsquigarrow f^r) = \frac{count(f, f^r)}{\sum_{f' \in perm(f)} count(f, f')} \tag{3}$$

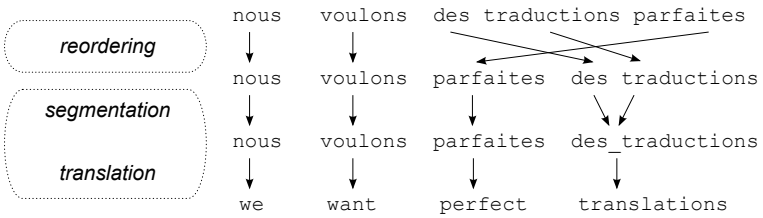
where f and f^r are the original (or left-hand side) and reordered (right-hand side) sequence of source words (or factors) of the considered rewrite rule, and $perm(f)$ is the set of permutations of f.

Lexicalized reordering NCODE implements the standard lexicalized reordering (Tillman, 2004) with four basic reordering types: **(m)**onotone order; **(s)**wap with previous tuple; **(f)**orward jump; **(b)**ackward jump. In addition, we also consider two aggregated types: **(d)**iscontinuous, corresponding to (b) and (f) and finally **(c)**ontinuous, corresponding to (m) and (s). In order to estimate these models, we count how often each tuple is found with each of the four orientation types ($orientation \in \{m, s, f, b\}$), and used the following smoothed maximum likelihood estimators ($\sigma = 1/\sum_o count(o, f, e)$):

$$P(orientation|f, e) = \frac{(\sigma/4) + count(orientation, f, e)}{\sigma + \sum_o count(o, f, e)} \tag{4}$$

Source-reordered n-gram LM N-gram language models estimated over the source words (or factors) of the training corpus are also estimated. Note that the training source words are first reordered following the tuple extraction process. The model scores a given source-side reordering hypothesis according to the reorderings performed in the training sentences following the word alignments. Again, the model is estimated as any standard n-gram language model, use `--options-sm` and `--name-src-unf` respectively to set the language model options and to identify the potentially multiple language model files. See (Crego and Yvon, 2009) for details.

Figure 2. Decoding with NCODE.



4. Decoding

Figure 2 details the various processing steps taken in our system: source words are first shuffled in various ways (*reordering*) so as to reproduce the target word order; source sentences are then segmented (*segmentation*); translations of each segment (*translation*) are produced in the last step. The final translation hypothesis is obtained by concatenation of such partial hypotheses. Note that multiple choices are considered at each step, defining the *decoding search space*.

As introduced in Section 1, our toolkit splits the reordering and decoding problems into separate modules (detailed below), aiming at better tackling each of the problems. The first module computes *reordering* hypotheses producing a word permutation lattice. The word lattice is then traversed in the second step, where *segmentation* and *translation* take place. An intermediate step is introduced after the reordering module, which only keeps those units than are useful to translate the input sentence.

Permutation lattice and test filtering Sentences to be translated are encoded as word lattices (use `binrules`) containing the most promising reordering hypotheses, so as to reproduce the word order modifications introduced during the tuple extraction process. Hence, reordering rules are applied on top of the input sentences to be translated. More formally, given an input sentence, f, in the form of a linear word automaton, and N optional reordering rules to be applied on f, each of which is represented by a finite-state transducer τ_i, the resulting reordering lattice f^* is obtained by the sequential composition of FSTs, as:

$$f^* = \tau_N \circ \tau_{N-1} \cdots \circ \cdots \tau_1 \circ f$$

where $\circ$ denotes the composition operation. Note that the sequence of FSTs is sorted according to the length of the left-hand side (LHS) of the rule. More specific rules, having a larger LHS, are applied (composed) first, in order to ensure the recursive application of the rules. Hence, some paths are obtained by applying reordering on top of already reordered paths. The applied rules can be limited to a maximum size of words (use `-maxr`) and to a maximum cost, or negative log of the probability estimated according to Equation 3 (use `-maxc`).

Once the word lattice is computed, the tuple vocabulary is then filtered for each input sentence (use `binfiltr`), removing all tuples but those units whose source-side matches any of the n-grams appearing in the lattice. The resulting file contains all the modeling information needed by the decoder to translate sentences, with the exception of n-gram language models scores. Tuples (source and target) size can be limited to a given number of words (use `-maxs`).

Search As for any statistical MT system, translation are built by searching a vast (theoretically infinite) set of translation hypotheses. Search stops when the most likely hypothesis covering (translating) the entire input sentence is attained (use `bincoder`). The algorithm is slightly modified to output n-best (use `-nbest`) hypotheses or the complete search graph (use `-ograph`).

As in most cases an exhaustive search is unfeasible pruning techniques are required, aiming at discarding partial hypotheses based on (more or less fair) hypotheses comparisons strategies. For NCODE, partial hypotheses are spread over multiple stacks[6] according to the source words they translate (use `-s 2J`). Each stack contains hypotheses that translate strictly the *same source words*.

A well-known heuristic technique then consists in discarding the worst ranked hypotheses of each stack. This idea can be implemented in several ways, the most common being known as *beam pruning* (use `-b i`), which expands the subset of the i-*best stack hypotheses*. Another common practice consists in considering only the i-*best translation choices* for each source segment (use `-t i`), what provides additional computational savings, but typically yields crudest heuristics, as the pruning is only based on non-contextualized translation scores. Hypotheses *recombination* is also implemented by NCODE, another risk-free way to discard hypotheses. However, the decoder is very often interested in the translation search graph, for which hypotheses recombinations need to be carefully recorded rather than discarded.

This organization of the search space ensures that, within a stack, hypotheses can be compared on a fair basis; this is at the cost, however, of inefficiencies when the size of the input lattice increases. As each node in this lattice corresponds to one stack of the decoder, we would need up to 2^J stacks to process a lattice encoding all the permutations of a sentence of length J. An alternative is organize stacks based simply on the *number of target words* (use `-s J`). This solution is more efficient, yet, it may bias the search towards translating first the easiest parts of the source sentence. This bias can be reduced using estimations of future costs, a workaround that has not yet been implemented, due to the complexity of accurately computing these estimations in our architecture.

Finally note that a simple way to increase efficiency when translating multiple input sentences over multi-threaded architectures consists of running on several

[6] More precisely: priority lists. We prefered to stick to the usual MT terminology here.

threads (use `-threads i`). Up to `i` sentences, depending on the server architecture and availability, will be translated *'in parallel'*.
Ncode's default policy to handle out-of-vocabulary words (source words never seen in the training text) is to output the source (unknown) word in the translation stream. An alternative option is to drop the unknown word (use `-dropunk`). The set of translation units used in the o*n*e-best translation option can be produced (use `-units`). A verbose mode (use `-verbose`) is also available which produces an *extremely detailed* output of the search process.

5. Tuning the Set of Models

As explained above, Ncode scores hypotheses based on a linear combination of several model scores. In order to obtain accurate translation results, the contribution of each model in the translation process needs to be tuned. Such optimization is carried out in our toolkit by the script `mert-run.perl`, which merely serves as wrapper for the MERT toolkit[7] implemented in the Moses toolkit. Once the optimal set of model weights is found, the script `mert-tst.perl` performs the translation of test sentences, using the models and configuration used in the optimization process, together with the optimized set of model weights.

6. A Comparison with Moses

In this section, we compare the performance of Ncode with that of Moses (Koehn et al., 2007), a state-of-the-art SMT system for phrase-based translation models.

The systems are compared on two different French-to-German translation tasks. The first (**news**) is a *small* size data task considering the *News Commentary* bitext made available in the *Sixth Workshop on Statistical Machine Translation*[8]. A second task (**full**) includes additional training data, ending up with a bitext of near four million sentences. Development work (tuning) is carried out for both systems using the *newstest2010* test set. Performance is evaluated over the *newstest2009* and *newstest2011* test sets available for the same translation task.

Thus, both approaches are compared using the same training corpora, target language model and word alignments, obtained performing Giza++[9]. The TreeTagger[10] toolkit was used to obtain the POS tags needed by Ncode. Afterwards, a default configuration of both toolkits is also used to perform training, tuning and decoding. Moses performs translation using the default 14 scores, while Ncode employs 2 bilingual *n*-gram language models. The first is estimated over tuples built from surface

[7] http://www.statmt.org/moses/?n=FactoredTraining.Tuning

[8] http://www.statmt.org/wmt11/translation-task.html

[9] http://www.fjoch.com/GIZA++.html

[10] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

Table 1. Performance statistics measured for Ncode *and* Moses *SMT systems.*

System	*Task*	*BLEU*		*#units*	*Speed*	*Memory*
		newstest2009	*newstest2011*			
Ncode	*news*	13.89	13.83	0.5	54.4	7.7
	full	15.09	15.26	7.5	33.9	9
Moses	*news*	13.70	13.51	7.5	23.1	7.9
	full	14.66	14.51	141	14.7	16

word forms, the second with tuples built from POS tags in both sides. Table 1 details performance measurements obtained for both systems. Translation accuracy is measured by the *BLEU* score (Papineni et al., 2002). The total number of units (in millions) obtained after training (tuples and phrases are limited to a maximum of 6 words) is displayed in the fifth column. Translation speed is reported in terms of words per second in the sixth column. Finally, the last column contains an approximation of the memory (in Mb) used by each decoder.

As can be seen, translation accuracy results are slightly higher for Ncode in both translation tasks and test sets, although all differences fall within the statistical confidence margin (add ± 1.50 BLEU for a 95% confidence level). In terms of data efficiency, Ncode clearly outperforms Moses: a unique segmentation is used to collect tuples, yielding a much smaller set of tuples than of phrases. In the case of the **full** task, the vocabulary of phrases is 20 times larger than the corresponding set of tuples. Ncode also outperforms Moses when considering the amount of memory needed by the decoder. Ncode needs about half of the memory needed by Moses (**full** data task). Notice that in the case of the **small** data task, the difference in the amount of memory needed is very small. This can be explained by the fact that both vocabularies of translation units are very small compared to the target language model, which account for most of the memory needs of both decoders. According to translation speed, measures were taken performing single-threaded translations by both decoders. Results show that Ncode is nearly twice faster than Moses.

7. Conclusions

We have described Ncode, an open source statistical machine translation toolkit for translation models estimated as n-gram language models. It can be downloaded from http://ncode.limsi.fr. We reviewed the main features that are currently implemented. Additionally, we carried out a short comparison with the widely known Moses SMT system. Ncode showed slightly higher French-to-German translation accuracy results than Moses. Our decoder also outperformed Moses in terms of memory requirements and translation speed.

Acknowledgements

This work was partially funded by OSEO under the Quaero program. The authors also want to thank those researchers of the UPC (Adrià de Gispert, Marta Ruiz, Patrik Lambert) and LIMSI (Alexandre Allauzen, Aurélien Max, Thomas Lavergne, Artem Sokolov) who contributed to create and extend the toolkit.

Bibliography

Casacuberta, F. and E. Vidal. Machine translation with inferred stochastic finite-state transducers. Computational Linguistics 30(4):205–225, 2004.

Crego, Josep M. and José B. Mariño. Improving SMT by coupling reordering and decoding. Machine Translation, 20(3):199–215, 2007.

Crego, Josep M. and Franc cois Yvon. Improving reordering with linguistically informed bilingual n-grams, August 2009.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), demonstration session, Prague, Czech Republic, June, 2007.

Mariño, José B., Rafael E. Banchs, Josep M. Crego, Adrià de Gispert, Patrick Lambert, José A.R. Fonollosa, and Marta R. Costa-Jussà. N-gram-based machine translation. Computational Linguistics 32(4):7–549, 2006.

Och, Franz Josef and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), 295–302, Philadelphia, PA, July, 2002.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation, 2002.

Tillman, Christoph. A unigram orientation model for statistical machine translation. Proceedings of the HLT-NAACL'04, 101–104, Boston, MA, USA, May, 2004.

Zens, Richard, Franz Joseph Och, and Herman Ney. Phrase-based statistical machine translation. Proceedings of the 25th Annual German Conference on AI: Advances in Artificial Intelligence, 18–32, 2002.

Address for correspondence:
Josep M. Crego
jmcrego@limsi.fr
LIMSI-CNRS, BP 133, 91430 Orsay Cedex (France)