PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 17–26

Memory-Based Machine Translation and Language Modeling

Antal van den Bosch, Peter Berck

Abstract

We describe a freely available open source memory-based machine translation system, MBMT. Its translation model is a fast approximate memory-based classifier, trained to map trigrams of source-language words onto trigrams of target-language words. In a second decoding step, the predicted trigrams are rearranged according to their overlap, and candidate output sequences are ranked according to a memory-based language model. We report on the scaling abilities of the memory-based approach, observing fast training and testing times, and linear scaling behavior in speed and memory costs. The system is released as an open source software package[1], for which we provide a first reference guide.

1. Introduction

Recently, several independent proposals have been formulated to integrate discrete classifiers in phrase-based statistical machine translation, to filter the generation of output phrases (Bangalore, Haffner, and Kanthak, 2007, Carpuat and Wu, 2007, Giménez and Màrquez, 2007, Stroppa, Van den Bosch, and Way, 2007), all reporting positive effects. This development appears an interesting step in the further development of statistical machine translation. These same developments can also be employed to produce simple but efficient stand-alone translation models.

In this paper, we introduce MBMT, memory-based machine translation. The memory-based approach, based on the idea that new instances of a task can be solved by analogy to similar instances of the task seen earlier in training and stored in memory as such, has been used successfully before in various NLP areas; for an overview, see (Daelemans and Van den Bosch, 2005).

MBMT is a stand-alone translation model with a simple decoder on top that relies on a memory-based language model. With a statistical word alignment as the starting point, such

[1] http://ilk.uvt.nl/mbmt

Please cite this article as: Antal van den Bosch, Peter Berck, Memory-Based Machine Translation and Language Modeling. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 17–26.

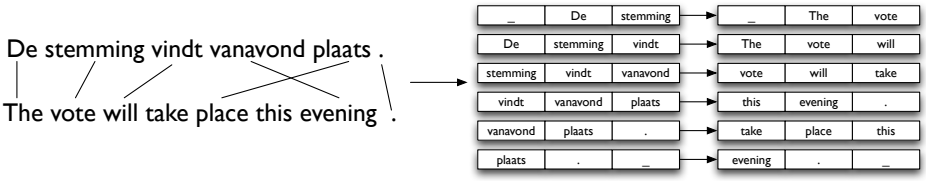

Figure 1. An example training pair of sentences, converted into six overlapping trigrams with their aligned trigram translations.

as produced by GIZA++ (Och and Ney, 2003), MBMT is shown to be very fast both in training and translation.

The overall architecture of the system is described in Section 2. A brief evaluation and reference guide of the system is provided in Section 3. The language modeling module, also made available as a separate language modeling toolkit, is described in Section 4. We wrap up in Section 5.

2. Memory-based machine translation

Memory-based machine translation (Van den Bosch, Stroppa, and Way, 2007) can be characterized as an instantiation of example-based machine translation (EBMT), as it essentially follows EBMT's basic steps (Carl and Way, 2003): given a sentence in the source language to be translated, it searches the source side of the corpus for close matches and their equivalent target language translations. Then, it identifies useful source–target fragments contained in those retrieved examples; and finally, it recombines relevant target language fragments to derive a translation of the input sentence.

The scope of the matching function implied in the first step is an important choice. We take a simplistic approach that assumes no linguistic knowledge; we use overlapping trigrams of words as the working units, both at the source language side and the target language side.

The process of translating a new sentence is divided into a local phase (corresponding to the first two steps in the EBMT process) in which memory-based translation of source trigrams to target trigrams takes place, and a global phase (corresponding to the third EBMT step) in which a translation of a sentence is assembled from the local predictions. We describe the two phases in the following two subsections.

2.1. Local classification

Both in training and in actual translation, when a new sentence in the source language is presented as input, it is first converted into windowed trigrams, where each token is taken as the center of a trigram once. The first trigram of the sentence contains an empty left element, and the last trigram contains an empty right element. At training time, each source language sentence is accompanied by a target language translation. We assume that word alignment has

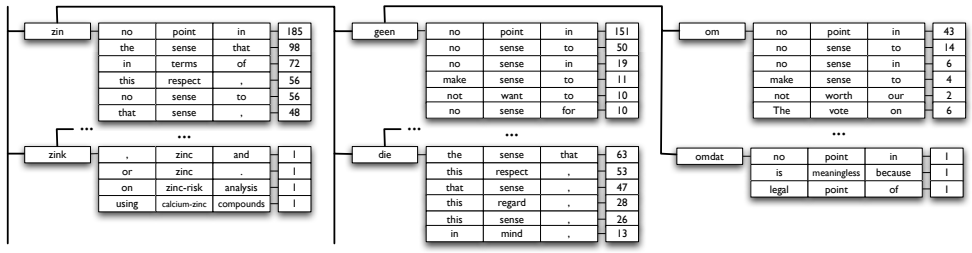

Figure 2. Excerpt of an mbmt igtree *structure, zooming in on the path represented by the input trigram "geen zin om", translatable to "no point in", among others.*

taken place, so that we know for each source word whether it maps to a target word, and if so, to which. Examples are only generated for source words that align to target words. Given the alignment, each source trigram is mapped to a target trigram of which the middle word is the target word to which the word in the middle of the source trigram aligns. The left and right neighboring words of the target trigram are the center word's actual neighbors in the target sentence. Figure 1 exemplifies the conversion of a training translation to six trigram mappings.

During translation, source trigrams are matched against the training set of stored source trigrams with a known mapping to a target trigram. The matching is carried out as a discrete classification. To this purpose we make use of IGTREE[2] (Daelemans, Van den Bosch, and Weijters, 1997), which compresses a database of labeled examples into a lossless-compression decision-tree structure that preserves the labeling information of all training examples. Figure 2 displays a fragment of the decision tree trained on the translation of Dutch trigrams to English. It highlights one path in the tree, representing the Dutch trigram "geen zin om" (translatable, among others, into "no point in" and "no sense to"). The tree encodes all possible trigram translations of, respectively, the middle word "zin", the bigram "geen zin", and the full trigram. This order reflects the information-gain weights of the three words with respect to predicting the output class.

During translation, IGTREE's classification algorithm traverses the decision tree, matching the middle, left, and right words of each new trigram to a path in the tree. Two outcomes are possible: (1) IGTREE finds a complete matching path, upon which it returns the most probable output trigram; (2) IGTREE fails to match a value along the way, upon which it returns the most probable output trigram given the matching path so far. Instead of the most probable path, it is also possible for IGTREE to return the full distribution of possible trigrams at the end of a matching path.

When translating new text, trigram outputs are generated for all words in each new source language sentence to be translated, since our system does not have clues as to which words would be aligned by statistical word alignment.

[2] `http://ilk.uvt.nl/timbl`

2.2. Global search

To convert the set of generated target trigrams into a full sentence translation, the overlap between the predicted trigrams is exploited. Figure 3 illustrates a perfect case of a resolution of the overlap (drawing on the example of Figure 1), causing words in the English sentence to change position with respect to their aligned Dutch counterparts. The first three English trigrams align one-to-one with the first three Dutch words. The fourth predicted English trigram, however, overlaps to its left with the fifth predicted trigram, in one position, and overlaps in two positions to the right with the sixth predicted trigram, suggesting that this part of the English sentence is positioned at the end. Note that in this example, the "fertility" words *take* and *this*, which are not aligned in the training trigram mappings (cf. Figure 1), play key roles in establishing trigram overlap.

In contrast to the ideal situation sketched in Figure 3, where one translation is produced, in practice many different candidate output sequences can be generated due to two reasons: first, each (potentially partially or fully incorrect) trigram may overlap with more than one trigram to the left or right, and second, the classifier may produce more than one output trigram at a single position, when it reaches a non-ending node with equally-probable trigram classes.

To select the most likely output among the potentially large pool of candidate outputs, we employ a memory-based target language model (Van den Bosch, 2006). This model, called WOPR, described in more detail in Section 4, is a word prediction IGTREE system trained on a monolingual target language corpus, which produces perplexity scores for each candidate output sequence presented to it.

WOPR provides the language model in a different way than most standard models do. Most models, like for example SRILM (Stolcke, 2002), estimate probabilities of words in context and build a back-off model containing n-grams to unigrams. WOPR uses a trigram model (it is not limited to trigrams, it could use any size and any context) but, because it uses the IGTREE algorithm, stores exceptions to default values rather than all n-grams. Other language models could be used, but we prefer to use WOPR because it uses the same IGTREE model also used as the core translation engine in the MBMT system. The model is described in more detail in Section 4.

As the number of possible output sequences may be large, MBMT currently applies Monte Carlo sampling to generate candidate output sequences to be scored by WOPR. This sampling is subject to a patience threshold p that halts the generation of new candidates when no improvement in perplexity scores is observed for p sample steps. By default, $p = 100$.

3. Evaluation and an Annotated Example

For the purpose of a brief evaluation, we first focus on the translation of Dutch to English, using the EMEA corpus, part of the Opus open source parallel corpus[3]. The EMEA corpus contains documents from the European Medicines Agency[4]. Texts in this corpus are of a re-

[3] `http://urd.let.rug.nl/tiedeman/OPUS/` – downloaded in June 2008.

[4] `http://www.emea.europa.eu/`

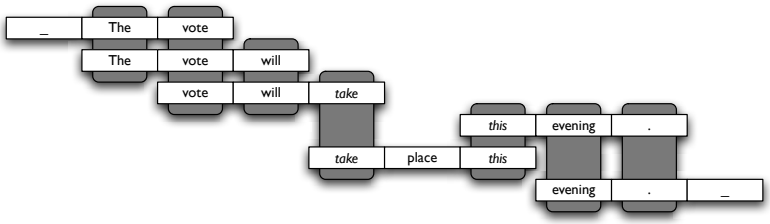

Figure 3. Producing a global solution by resolving the overlap between six trigrams. Italicized words are not aligned with source words.

stricted genre, consisting of quite formal, exact, and largely controlled language. We used the first 749,602 lines of text (approximately 9.0 million English and Dutch words). The corpus was split into a final 1,000-sentence test set and a training set containing the remainder of the data. The training sets were word-aligned using the GIZA++ algorithm (Och and Ney, 2003). No decapitalization was performed. The IGTREE-based language model used for translation is a single model trained on the first 112 million words of the Reuters RCV1 corpus.

We performed a learning curve experiment on the EMEA training set. We start at a training set size of 100,000 tokens, and increment with steps of 100,000 until 1 million tokens; then, we increment with steps of 1 million tokens up to the maximal training set size of 9 million tokens. The learning curve experiment serves to get an idea of the scaling abilities of MBMT in terms of performance; we also measure training and testing speeds and memory footprint. The learning curve experiment on the EMEA corpus produced performance curves of which we combine two in the left graph of Figure 4: the BLEU and METEOR (exact) scores. Both curves show a steady but somewhat weakening increase when the dataset doubles in size (note that the x axis is logarithmic).

Second, the middle graph of Figure 4 displays the number of seconds it takes to construct a decision tree, and to test. Testing occurs in a few seconds (up to eight seconds for 1,000 sentences, with an approximately linear increase of one second of testing time with each additional million training examples); the test graph virtually coincides with the x axis. Training times are more notable. The relation between training times and number of training examples appears to be linear; on average, each additional million of training examples makes training about 130 seconds slower.

Third, the right graph of Figure 4 shows a similar linear trend of the memory footprint needed by IGTREE and its decision tree, in terms of Megabytes. At 9 million training examples, the decision tree needs about 40 Mb, an average increase of 4.4 Mb per additional million examples.

As a second evaluation, we compare against the performance and training and testing times of MOSES on the EMEA corpus at the maximal training set size. Table 1 lists the performance on the test data according to word error rate, position-independent word error rate, BLEU, Meteor, and NIST. As is apparent from the results, MOSES performs at a markedly higher level

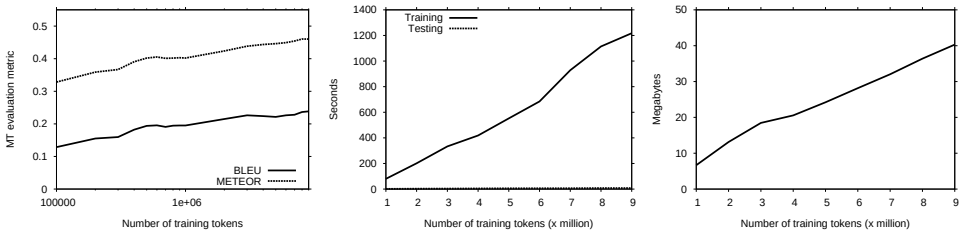

Figure 4. Learning curves on the emea *corpus in terms of MT evaluation (*bleu *and* meteor*, left, with a logarithmic x-axis), seconds to train and test (middle; the test graph virtually coincides with the x axis), and memory needed (right), with increasing amounts of training material available.*

of performance, but does so at the cost of a longer translation time: MBMT is about 20 times as fast. Training MBMT is about 10 times as fast as training MOSES; in both cases, the GIZA++ process has already been performed and is not included here.

System	WER	PER	BLEU	Meteor	NIST	Training (h:m:s)	Test (m:s)
MBMT	72.7	63.6	0.238	0.460	4.97	20:17	0:08
MOSES	46.6	39.4	0.470	0.650	7.06	3:10:06	2:51

Table 1. Comparing mbmt *agains* moses *on the* emea *corpus, in terms of five MT evaluation metrics, and training and testing times (elapsed wallclock time).*

Annotated Example

The MBMT software assumes a GIZA++-style A3 file, i.e. a word alignment file containing all aligned source and target training sentences, as training material. The software will convert this training file into an IGTREE decision tree, and is then capable of translating a raw text file in the source language (tokenized, one sentence per line) into a translated raw text file in the target language (also one sentence per line). The commandline functionality is currently limited to the identification of the A3 training file, and the source-language test text, plus the optional setting of the patience threshold to a non-default setting, e.g. $p = 50$, with the `-p` switch:

```
mbmt -t EMEA.9m.train.A3.final -t EMEA-dutch.test.txt -p50
```

During runtime, MBMT generates several intermediary files. First, the A3 file is converted to a training file suited for IGTREE, mapping source-language trigrams to aligning target-language trigrams (cf. Figure 1). Subsequently, this file is compressed into an IGTREE decision tree, at

a typical compression rate of about 95%. The test set is also converted into trigram instances (one instance per word), which are then classified by IGTREE. This output is stored in a file of which the first line looks as follows:

```
na de behandeling ? after_the_end { after_the_end 3.00000,
,_the_rate 3.00000, after_the_treatment 3.00000 }
```

The Dutch trigram *na de behandeling* ("after the treatment") is classified by IGTREE as mapping to three equally likely trigram translations. These three translations will be carried along to the final phase, where all predicted trigrams are used to generate possible translations, using a Monte Carlo sampling method with a halting criterion governed by the patience parameter p. Each candidate output sequence is scored by WOPR, the memory-based language model.

4. Memory-based language modeling

The MBMT system generates a candidate number of translations for each input sentence. The typical approximate solution to picking the best translation is to use a language model to determine which translation fits best in the target language, e.g. selecting the candidate string with the lowest perplexity score.

In the MBMT system, WOPR is the language model. It is a *word predictor* based on IGTREE, trained to predict the next word in a sentence (Van den Bosch, 2006). To calculate the perplexity of a sentence, we feed it to WOPR and see which words it predicts for each word in the sentence. The perplexity is calculated from the estimated probabilities of each prediction. A prediction is a classification by IGTREE based on a local context of preceding words. In contrast with how IGTREE is used in the MBMT translation module, the word predictor classifier in WOPR produces class distributions (with more than one class if the classification occurs at a non-ending node).

Thus, WOPR usually returns more than one word for a given sequence, together with a probability based on frequency counts. This distribution of possible answers is used to calculate a perplexity value. There are three possibilities: (1) If the distribution returned by IGTREE contains the correct word, we take the probability of the word in the distribution; (2) If the distribution does not contain the correct word, we check if it is in the lexicon. If it is, the lexical probability is taken; (3) If it is not in the lexicon, a probability for unseen items is used that is estimated through Good-Turing smoothing. WOPR calculates the sum of $-p \log_2(p)$ of all the probabilities (one for each word in the sentence), and divides this by the number of words to obtain the average over the sentence. The perplexity value is two to the power of this sum.

Annotated Example

Besides its language modeling functionalities of predicting words and measuring perplexities, WOPR also provides the necessary tools to prepare the data, create datasets and train its prediction models. The following shows how a memory-based language model can be created starting from plain text data. Let us assume the file is called `corpus1.txt`. WOPR commands

generally have two parameters. The first one, `-r` tells WOPR which subroutine or tool to run. The second parameter, `-p` is a comma separated list of `keyword: value` pairs which specify the different parameters.

As a first step, WOPR is used to create a lexicon, which in this case is a list of words and their frequency in the corpus. It also generates a list with "counts of counts", which is used in Good-Turing smoothing of probabilities.

```
wopr -r lexicon -p filename: corpus1.txt
```

WOPR creates output file names based on the input file name and the command that is executed. In this case, it creates two files called `corpus1.txt.lex` and `corpus1.txt.cnt`. Next, we generate our windowed dataset. In this example we use a window size of three previous words. The resulting file is called `corpus1.txt.ws3`.

```
wopr -r window_s -p filename: corpus1.txt, ws: 3
```

We want to discard words with a frequency of five or less from our data set, and replace them with a special token <unk>. This is done with the following command:

```
wopr -r hapax -p filename: corpus1.txt.ws3, lexicon: corpus1.txt.lex,
hpx: 5
```

We then train our instance base. WOPR is used as a wrapper in this case, and most of the work is done by IGTREE. This could take some time, depending on the size of the data, but once the decision tree has been created and saved, it can easily be read in and used again.

```
wopr -r make_ibase -p corpus1.txt.ws3.hpx5, timbl: "-a1 +D"
```

Now we are ready to run our word predictor on a test file. The command to do this is as follows:

```
wopr -r pplxs -p filename: test1.txt.ws3.hpx5, ibasefile:
corpus1.txt.ws3.hpx5.ibase, timbl: "-a1 +D"
```

The test data is prepared in the same way as the training data. The following shows a line of output from WOPR. It shows an instance (`I would like`), the following word (`to`) and the –in this case correct– guess from the predictor, `to`.

```
I would like to to -0.351675 1.8461 1.27604 65 [ to 768 the 34
a 20 it 12 an 12 ]
```

This is followed by a number of statistics. The logprob of the prediction is -0.351675. The entropy of the distribution returned by IGTREE is 1.8461 ($-\sum p \log_2(p)$). The third number

shows the word level perplexity ($2^{-\mathrm{logprob}}$). The last number shows the number of elements in the distribution, in this case 65. This is followed by a top 5 of the distribution returned (with counts).

It is also possible to run WOPR in server mode, communicating over a socket connection with a client; in fact, this is how it is incorporated in the MBMT system. In server mode, WOPR will wait for a connection by another program and process the data it receives. The answer is sent back over the same connection.

5. Discussion

We have released MBMT, a straightforward translation model based on a fast approximation of memory-based classification. The approach fits into the EBMT framework; it models the mapping of sequences of word spans (here, word trigrams) in the source language to word trigrams in the output language. We showed that MBMT scales well to increased amounts of learning material. Within the current experiments we observed that training time and memory storage costs are approximately linear in the number of training examples. Translation speed on unseen data is very fast; our test set of 1,000 sentences was processed within seconds. Based on these results, we conclude for now that memory-based machine translation systems may be relevant in cases in which there is a need for fast and memory-lean training and/or classification. The low memory footprint may be additionally interesting for implementations of such systems in limited-capacity devices.

As a separate component of MBMT we have released the memory-based language model software package WOPR, which can also be used in isolation for general language modeling purposes. WOPR offers its functionality through command line options, but can also run in server mode; this is how MBMT uses WOPR.

Acknowledgments

This research is funded by NWO, the Netherlands Organisation for Scientific Research. We are grateful to Nicolas Stroppa, Andy Way, and Patrik Lambert for discussions and help with the comparison with Moses.

Bibliography

Bangalore, S., P. Haffner, and S. Kanthak. 2007. Statistical machine translation through global lexical selection and sentence reconstruction. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 152–159, Prague, Czech Republic. Association for Computational Linguistics.

Carl, M. and A. Way. 2003. *Recent Advances in Example-Based Machine Translation*, volume 21 of *Text, Speech and Language Technology*. Dordrecht, the Netherlands: Kluwer.

Carpuat, M. and D. Wu. 2007. Improving statistical machine translation using word sense disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 61–72.

Daelemans, W. and A. Van den Bosch. 2005. *Memory-based language processing*. Cambridge University Press, Cambridge, UK.

Daelemans, W., A. Van den Bosch, and A. Weijters. 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

Giménez, J. and L. Màrquez. 2007. Context-aware discriminative phrase selection for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 159–166, Prague, Czech Republic, June. Association for Computational Linguistics.

Och, F.-J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Stolcke, A. 2002. SRILM – An extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904, Denver, CO.

Stroppa, N., A. Van den Bosch, and A. Way. 2007. Exploiting source similarity for SMT using context-informed features. In A. Way and B. Gawronska, editors, *Proceedings of the 11th International Conference on Theoretical Issues in Machine Translation (TMI 2007)*, pages 231–240, Skövde, Sweden.

Van den Bosch, A. 2006. Scalable classification-based word prediction and confusible correction. *Traitement Automatique des Langues*, 46(2):39–63.

Van den Bosch, A., N. Stroppa, and A. Way. 2007. A memory-based classification approach to marker-based EBMT. In F. Van Eynde, V. Vandeghinste, and I. Schuurman, editors, *Proceedings of the METIS-II Workshop on New Approaches to Machine Translation*, pages 63–72, Leuven, Belgium.