PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 93 JANUARY 2010 27–36

Combining Machine Translation Output with Open Source
The Carnegie Mellon Multi-Engine Machine Translation Scheme

Kenneth Heafield, Alon Lavie

Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213

Abstract

The Carnegie Mellon multi-engine machine translation software merges output from several machine translation systems into a single improved translation. This improvement is significant: in the recent NIST MT09 evaluation, the combined Arabic-English output scored 5.22 BLEU points higher than the best individual system. Concurrent with this paper, we release the source code behind this result consisting of a recombining beam search decoder, the combination search space and features, and several accessories. Here we describe how the released software works and its use.

1. Introduction

Research in machine translation has led to many different translation systems, each with strengths and weaknesses. System combination exploits these differences to obtain improved output. Many approaches to system combination exist; here we discuss an improved version of (Heafield et al., 2009) that, unlike most other approaches, synthesizes new word orderings. Since September 2008, the code we release has been completely rewritten in multithreaded C++ that produces 2.9 combined translations per second. Along with the core system combination code, we also release language modeling and evaluation tools of use to the machine translation community. All of these are available for download at `http://kheafield.com/code/mt/`.

The scheme has several parts. Hypotheses are aligned in pairs using the publicly available METEOR (Banerjee and Lavie, 2005) aligner. A search space (Heafield et al., 2009) is defined on top of these alignments. Beam search is used to make this search tractable. Recombination increases efficiency and diversity by packing hypotheses

© 2010 PBML. All rights reserved.
Corresponding author: heafield@cs.cmu.edu
Cite as: Kenneth Heafield, Alon Lavie. Combining Machine Translation Output with Open Source: The Carnegie Mellon Multi-Engine Machine Translation Scheme. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 27–36. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0008-4.

that extend in the same way. Hypotheses are scored using a linear model that combines a battery of features detailed in Section 3. Model weights are tuned using Z-MERT (Zaidan, 2009).

The remainder of this paper is organized as follows. Section 2 surveys other system combination techniques. In Section 3 we describe the components of the system with reference to code while Section 4 shows how to run the system. Section 5 summarizes results in recent evaluations and Section 6 concludes.

2. Related Work

Confusion networks (Rosti et al., 2008; Karakos et al., 2008) are a popular form of system combination. This approach combines k-best output from multiple systems. A single k-best list entry is selected as the backbone, which determines word order. The backbone may be selected greedily using some agreement metric or jointly with the full decoding problem (Leusch et al., 2009). Once the backbone is selected, every other k-best entry is aligned to the backbone using exact matches and position information. Translation Edit Rate (Snover et al., 2006) is commonly used for this purpose, with the substitution operation corresponding to alignment. This alignment is still incomplete; unaligned words are aligned to the empty word, corresponding to the deletion (if in the backbone) or insertion (if in a different sentence) operations of TER. Within each alignment, entries vote on word substitution, including with the empty word. Selection of the backbone and word substitution are the only options considered by confusion networks.

The next type of system combination jointly resolves word order and lexical choice. In our approach, we permit the backbone to switch as often as each word. Closely related work (He and Toutanova, 2009) uses a reordering model like Moses (Koehn et al., 2007) to determine word order. While they resolve ambiguous position-based alignments jointly with decoding, we use METEOR to greedily resolve ambiguities resulting from knowledge-based alignments. Since these approaches allow many new word orders, both employ features to control word order by counting n-gram agreement between the system outputs and candidate combination. We use jointly tunable n-gram and system weights for these features; other work uses tunable system weights for at most unigrams (Zhao and He, 2009).

3. Components

3.1. Alignment

Rather than align to a single backbone, we treat single best outputs from each system symmetrically. All pairs are aligned using METEOR. It identifies, in decreasing order of priority:

1. Case-insensitive exact matches

2. Snowball (Porter, 2001) stem matches
3. Shared WordNet (Fellbaum, 1998) synonyms
4. Unigram paraphrases from the TERp (Snover et al., 2008) database

By contrast, confusion networks typically stop with exact matches and use position-based techniques to generate additional alignments. We eschew position-based methods since they can align content words with function words, leading to dropped content not noticed by BLEU (Karakos, 2009). In fact, we replaced the position-based artificial alignments of (Heafield et al., 2009) with the paraphrase database, finding similar performance. The `MEMT/Alignment` directory contains a Java class that calls the publicly available METEOR code to perform pairwise alignments. Since METEOR includes the WordNet database and a tool to extract the paraphrases, neither WordNet nor TERp is required.

3.2. Search Space

The search space is defined on top of the aligned sentences. A hypothesis starts with the first word of some sentence. It can continue to follow that sentence, or can switch to following a different sentence after any word. What results is a hypothesis that weaves together parts of several system outputs. In doing so, we must ensure that pieces cover the sentence without duplication and are fluent across switches. Duplication is prevented by ensuring that a hypothesis contains at most one word from each group of aligned words. A hypothesis may only switch to the first unused word from another output, thereby ensuring that the hypothesis covers the entire sentence. However, this can sometimes be too conservative, so a heuristic permits skipping over words in some cases (Heafield et al., 2009). That paper introduced two choices of heuristic and a radius parameter; here we use the length heuristic with radius 5. Code for the search space appears in the `MEMT/Strategy` directory. We use features to reward fluency.

3.3. Features

Since the search space so easily switches between sentences, maintaining fluency is crucial. A number of features are used for scoring partial and complete hypotheses:

Length The hypothesis length, as in Moses (Koehn et al., 2007). This compensates for the linear impact of length on other features.

Language Model Log probability from a Suffix Array (Zhang and Vogel, 2006) or an ARPA format language model. These appear in the `lm` directory with a simple common interface. We avoid direct dependence on the SRI (Stolcke, 2002) toolkit by providing our own equivalent implementation of inference.

Backoff Average n-gram length found in the language model. This provides limited tunable control over backoff behavior.

Match For each small n and each system, the number of n-gram matches between the hypothesis and system.

Each feature class has a directory under `MEMT/Feature`. Features have a common interface designed to make adding additional features easy. All features are combined in a *linear* model, which is equivalent to a log-linear model with each feature exponentiated. Model weights, especially for the match features, are heavily dependent on the underlying systems. We therefore provide scripts in `MEMT/scripts/zmert` to tune weights using Z-MERT (Zaidan, 2009). With 20 or more features, the optimization part of each iteration typically takes longer than does decoding.

3.4. Beam Search

Since the search space is exponential in the sentence length, we use beam search with recombination. The beam contains a configurable number of hypotheses of equal length; we typically keep 500 hypotheses. In order to increase beam diversity and speed decoding, we recombine hypotheses that will extend in the same way and score proportionally. Hypotheses to recombine are detected by hashing the search space state, feature state, and hypothesis history up to a length requested by the features. Recombined hypotheses are packed into a single hypothesis that maintains pointers to the packed hypotheses. At the end of the sentence, these packed hypotheses comprise a lattice where each node is labeled with the maximum-score path back to the beginning of the sentence. This enables efficient k-best extraction. The beam search decoder is factored into `MEMT/Decoder`. It only knows about the search space and features via template arguments and, therefore, may be independently useful for other left-to-right beam search problems.

4. Running Combination

4.1. Requirements

We assume a UNIX environment with a C++ compiler, Java, and Python. Scripts are provided in `install/` to install Boost, Boost Jam, ICU, and Ruby without requiring root access. Compiling consists of running `bjam release` in the `MEMT` directory. See the `README` file for more information.

A separate tuning set is required to learn parameter weights. This should be held out from system training or tuning data. We recommend reserving at least 400 segments for this purpose. A language model is also required; many use the SRILM toolkit (Stolcke, 2002) to produce ARPA files for this purpose. It should be tokenized the same way as the system outputs. A tokenizer is not provided; one can be downloaded from `http://www.statmt.org/wmt09/scripts.tgz` (Callison-Burch et al., 2009).

4.2. Alignment

The `MEMT/Alignment/MatcherMEMT.java` class uses the METEOR API to infer alignments. It should be compiled by running `MEMT/Alignment/compile.sh`. This script

will also download and install METEOR if necessary. Tokenized system outputs should be placed in text files with one segment per line. Running alignment is straightforward:

```
$ MEMT/Alignment/match.sh system1.txt system2.txt system3.txt >matched
```

4.3. Optional Language Model Filtering

Optionally, an ARPA language model can be filtered to the sentences being combined. The filter checks that an n-gram's vocabulary is a subset of some segment's vocabulary. This is much more strict than testing against the entire set's vocabulary, where words in an n-gram could be spread across several segments. The reduction in size can be dramatic: filtering a 19 GB ARPA file for the NIST MT09 Informal System Combination task produced a 1.4 GB ARPA file. Since the server will load this model into RAM, filtering greatly decreases hardware requirements. The command to filter to any number of matched files, including those with different sets of systems, is:

```
$ cat matched1 matched2 matched3 | MEMT/dist/FilterLM in.arpa out.arpa
```

The filter is fast: it keeps only the vocabularies in memory and takes about 12 minutes to filter a 19 GB model. This language model filter is also available as a separate package that reads one segment vocabulary per line. While phrase table expansion reduces effectiveness for statistical machine translation systems, we were still able to reduce model size by 36% by filtering to 1797 segments. It can also produce segment-level language model files if desired.

4.4. Decoding Server

The actual decoding algorithm runs inside a server process that accepts TCP connections. This avoids reloading the language model, which typically takes longer than performing thousands of combinations. The server is launched by specifying the language model and port:

```
$ MEMT/scripts/server.sh --lm.type ngram --lm.file lm.arpa --port 2000
```

When loading the language model has finished, it will print "Accepting Connections." Except for the language model and some threading options, configuration is sent by clients. Multiple connections with different configurations work properly. The protocol is highly compressible plain text, especially for k-best lists, so we advise using compressed SSH tunneling if the connection between client and server is slow.

4.5. Configuration

Most of the configuration options are set by clients of the decoding server. Figure 1 shows a configuration file without feature weights, which are added by tuning. Important hyperparameters to tweak are:

`horizon` The suboption `radius` controls how long words linger as described in Section 3.2. The `method` of distance measurement can be `length` or nearby `alignment`,

as described in (Heafield et al., 2009). Generally, a larger window works best in the presence of significant reordering. We recommend starting with `length` and a radius of 5.

`verbatim` Match features are called `verbatim` in the code. Two instances are provided; work on more flexible feature instantiation is planned. The idea behind two instances is that one does lexical voting using exact matches while the other uses all alignments to handle support and word order issues. The `mask` option controls which alignment types will count, including the implicit `self` alignment of words and `boundary` markers. The maximum match length to consider is also a key parameter. The `individual` option determines the maximum match length reported individually for each system. This may lead to too many features, so longer n-gram match counts can be presented on a `collective` basis by summing counts across systems.

`ouptut.nbest` Size of n-best output requested.

`length_normalize` This determines if feature values are divided by length, excepting of course the length feature itself. When disabled, the length feature otherwise acts to subtract the impact of length from other features. Empirically, we find turning off length normalization makes the output score slightly higher and output 1-2% longer.

Authoritative documentation of all options is printed when the server is run without an argument:

```
$ MEMT/scripts/server.sh
```

4.6. Tuning

Tuning requires a directory with three files: `decoder_config_base` containing the configuration file from Section 4.5, `dev.matched` containing the aligned tuning sentences from Section 4.2, and `dev.reference` containing the references (one per line). Multiple references for the same segment appear on consecutive lines. Assuming these files are in `work_dir` and the decoding server is running on port `2000`, the command line is:

```
$ MEMT/scripts/zmert/run.rb 2000 work_dir
```

If the server is running on another machine, it may be specified as `host:port`. This will run Z-MERT to tune the system and produce the file `work_dir/decoder_config` with tuned weights. It also decodes the tuning set with this configuration, placing output in `work_dir/output.1best`. Finally, it scores this tuning output against the provided reference, placing results in `work_dir/output.1best.scores`.

4.7. Decoding and Evaluation

Test data is decoded using the tuned configuration file and test `matched` file:

```
$ MEMT/scripts/simple_decode.rb 2000 decoder_config matched output
```

```
output.nbest = 300
beam_size = 500
length_normalize = false

#Remove words more than 5 behind as measured by length.
horizon.method = length
horizon.radius = 5

#Count exact matches up to length 2 for each system.
score.verbatim0.mask = "self exact boundary"
score.verbatim0.individual = 2
score.verbatim0.collective = 2
#Count non-paraphrase matches up to length 2 for each system.
#For length 3 and 4, sum the match counts across systems.
score.verbatim1.mask = "self exact boundary snowball_stem wn_synonymy"
score.verbatim1.individual = 2
score.verbatim1.collective = 4
```

Figure 1. Sample configuration file before tuning weights.

which creates `output.1best` with one segment per line and `output.nbest` in Moses (Koehn et al., 2007) format.

We provide a script that scores translations with BLEU (Papineni et al., 2002) from `mteval-13a.pl` (Peterson et al., 2009), NIST (Doddington, 2003), TER 0.7.25 (Snover et al., 2006), METEOR 1.0 (Banerjee and Lavie, 2005), unigram precision and recall, and length ratio. The following command generates the file `output.1best.scores` containing these respective scores:

```
$ Utilities/scoring/score.rb --hyp-tok output.1best --refs-laced ref
```

Running with `--print-header` will show column headers. Running without arguments provides the full list of options. This script is also available for download as a separate package.

5. Results

The 2009 Workshop on Machine Translation (WMT) (Callison-Burch et al., 2009) and NIST Open MT evaluations (Peterson et al., 2009) both added tracks specifically to evaluate system combination. We participated in both and now present updated unofficial results in Table 1. Gains on NIST data are surprisingly large–but not unexpected given the results from the evaluation (Peterson et al., 2009). Gains on WMT data depend mostly on the gap between Google and other systems; with a large gap, the effectiveness of system combination is minimal.

Source	System	BLEU	TER	METEOR
NIST Arabic	combo	58.55	36.86	70.76
	top single	*51.88*	*40.54*	*67.74*
NIST Urdu	combo	34.72	55.46	53.37
	top single	*32.88*	*56.20*	*52.24*
WMT Czech	combo	21.98	60.48	46.63
	top single	*21.18*	*59.57*	*46.91*
WMT French	combo	31.56	52.48	54.30
	top single	*31.14*	*51.36*	*54.91*
WMT German	combo	23.88	58.29	48.69
	top single	*21.31*	*60.78*	*56.82*
WMT Hungarian	combo	13.84	71.89	36.70
	top single	*12.75*	*68.35*	*35.43*
WMT Spanish	combo	28.79	53.63	53.51
	top single	*28.69*	*53.38*	*54.20*

Table 1. Unofficial post-evaluation scores on test data from past system combination tasks with the top system by BLEU shown in italics for comparison. The NIST MT09 Arabic-English scores are on unsequestered segments only. For 2009 Workshop on Machine Translation results, the language model is constrained; there was no constrained track for MT09 informal system combination. BLEU is uncased (and therefore not the official NIST MT09 metric), TER is version 0.7.25, and METEOR is version 1.0 with hter parameters.

6. Conclusion

We have released the source code to our system combination scheme. It shows significant improvement on some translation tasks, particularly those with systems close in performance. The software is ready to be downloaded, installed, and run. We hope to receive patches from users. In addition to the core system combination code, the language model filter and evaluation script are available as separate packages of general use to the community.

Acknowledgments

Funding for this work was provided by the DARPA GALE program and a National Science Foundation Graduate Research Fellowship. NIST serves to coordinate the NIST Open MT evaluations in order to support machine translation research and to help advance the state-of-the-art in machine translation technologies. NIST Open MT evaluations are not viewed as a competition, as such results reported by NIST are not to be construed, or represented, as endorsements of any participant's system, or as

official findings on the part of NIST or the U.S. Government. Informal System Combination was an informal, diagnostic MT09 task, offered after the official evaluation period.

Bibliography

Banerjee, Satanjeev and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, 2005.

Callison-Burch, Chris, Philipp Koehn, Christof Monz, and Josh Schroeder. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0401`.

Doddington, George. Automatic Evaluation of Machine Translation Quality Using N-gram Co-Occurrence Statistics. In *Proceedings of Human Language Technology Conference*, 2003.

Fellbaum, Christiane. *WordNet: An Electronic Lexical Database*. MIT Press, 1998. ISBN 978-0-262-06197-1.

He, Xiaodong and Kristina Toutanova. Joint optimization for machine translation system combination. In *EMNLP*, August 2009.

Heafield, Kenneth, Greg Hanneman, and Alon Lavie. Machine translation system combination with flexible word ordering. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 56–60, Athens, Greece, March 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W09/W09-0408`.

Karakos, Damianos. The JHU system combination scheme. In *NIST Open Machine Translation Evaluation Workshop*, Ottawa, Canada, September 2009.

Karakos, Damianos, Jason Eisner, Sanjeev Khudanpur, and Markus Dreyer. Machine translation system combination using ITG-based alignments. In *Proceedings ACL-08: HLT, Short Papers (Companion Volume)*, pages 81–84, 2008.

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. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic, June 2007.

Leusch, Gregor, Saša Hasan, Saab Mansour, Matthias Huck, and Hermann Ney. RWTH's system combination for the NIST 2009 MT ISC evaluation. In *NIST Open Machine Translation Evaluation Workshop*, Ottawa, Canada, September 2009.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, July 2002.

Peterson, Kay, Mark Przybocki, and Sébastien Bronsart. NIST 2009 open machine translation evaluation (MT09) official release of results, 2009. http://www.itl.nist.gov/iad/mig/tests/mt/2009/.

Porter, Martin. Snowball: A language for stemming algorithms, 2001. http://snowball.tartarus.org/texts/introduction.html.

Rosti, Antti-Veikko I., Bing Zhang, Spyros Matsoukas, and Richard Schwartz. Incremental hypothesis alignment for building confusion networks with application to machine translation system combination. In *Proceedings Third Workshop on Statistical Machine Translation*, pages 183–186, 2008.

Snover, Matthew, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings Seventh Conference of the Association for Machine Translation in the Americas*, pages 223–231, Cambridge, MA, August 2006.

Snover, Matthew, Nitin Madnani, Bonnie Dorr, and Richard Schwartz. TERp system description. In *Proceedings NIST Metrics MATR 2008*, 2008.

Stolcke, Andreas. SRILM - an extensible language modeling toolkit. In *Proc. ICSLP*, pages 901–904, 2002.

Zaidan, Omar. Z-MERT: A fully configurable open source tool for minimum error rate training of machine translation systems. *Prague Bulletin of Mathematical Linguistics*, 91:79–88, 2009.

Zhang, Ying and Stephan Vogel. Suffix array and its applications in empirical natural language processing. Technical Report CMU-LTI-06-010, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, December 2006.

Zhao, Yong and Xiaodong He. Using n-gram based features for machine translation system combination. In *Proceedings NAACL-HLT*, May 2009.