PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 91 JANUARY 2009 7–16

Improved Minimum Error Rate Training in Moses

Nicola Bertoldi, Barry Haddow, Jean-Baptiste Fouet

Abstract

We describe an open-source software for minimum error rate training (MERT) for statistical machine translation (SMT). This was implemented within the Moses toolkit, although it is essentially standsalone, with the aim of replacing the existing implementation with a cleaner, more flexible design, in order to facilitate further research in weight optimisation. A description of the design is given, as well as experiments to compare performance with the previous implementation and to demonstrate extensibility.

1. Introduction

1.1. Background

In Statistical Machine Translation (SMT), probabilistic models are used to find the best possible target translation $\mathbf{e}^*$ of a given source sentence $\mathbf{f}$, amongst all possible translations $\mathbf{e}$. The search for the best translation is known as *decoding*. The probabilistic models are estimated from bilingual and monolingual training data, and may include translation models, language models, reordering models, etc. In order to combine evidence from different models, it is standard practice to use a discriminative linear model, with the log probabilities as features. If the features of the model are $h_1, \ldots, h_r$, which depend on $\mathbf{e}$ and $\mathbf{f}$, then the best translation is given by

$$\mathbf{e}^*(\lambda) = \arg\max_{\mathbf{e}} \sum_{i=1}^{r} \lambda_i h_i(\mathbf{e}, \mathbf{f})$$

and depends on the feature weights $\lambda_1, \ldots, \lambda_r$.

The advantage of combining log probabilities in such a linear model is that the features can be arbitrary functions of the source and target sentences, and are not limited to just being log probabilities. For instance, a word count feature can be added which will penalize long or short sentences depending on the sign of the corresponding weight.

Please cite this article as: Nicola Bertoldi, Barry Haddow, Jean-Baptiste Fouet, Improved Minimum Error Rate Training in Moses. The Prague Bulletin of Mathematical Linguistics No. 91, 2009, 7–16.

The problem then arises of how to optimize the feature weights, in other words how to find a set of weights which will offer the best translation quality. The standard solution is to use minimum error rate training (MERT), a procedure introduced by Och (2003), which searches for weights minimizing a given error measure, or, equivalently, maximizing a given translation metric. This algorithm enables the weights to be optimized so that the decoder produces the best translations (according to some automatic metric Err and one or more references ref) on a development set of parallel sentences.

$$\lambda^* = \arg\min_{\lambda} \mathbf{Err}(\mathbf{e}^*(\lambda); \mathbf{ref})$$

The main feature of Och's approach is the exploitation of n-best translation alternatives (for each input sentence), that allows for fast convergence of the optimization process. The translation metric most commonly employed as the objective in MERT is the BLEU score (Papineni et al., 2001), although any automatic metric could in principle be used.

The weight optimization algorithm introduced in Och (2003) (and more fully described in Koehn (forthcoming)) is a form of coordinate ascent, where the search updates the feature weight which appears most likely to offer improvements, then iterates. Since calculation of the objective (in other words, the translation metric) is quite expensive, as much of it as possible is pre-calculated before running the optimization. Because the error surface is highly non-convex, MERT is always at risk of being trapped at local maxima; and because it uses n-best lists as an approximation for the decoder output, it cannot explore the actual parameter space. However, despite its limitations, MERT tends to produce good results.

MERT is the subject of ongoing research, for example to reduce the local maxima problem using regularization and stochastic search (Cer et al., 2008), to make convergence faster and more robust by selecting starting points by random walks (Moore and Quirk, 2008), and to replace the n-best lists by lattices (Macherey et al., 2008) and thereby improve the estimates of the expected translation score.

The MERT implementation discussed in this article is a subproject of Moses (Koehn et al., 2007), one of the leading open source implementations of phrase-based machine translation (Koehn et al., 2003). Having a state-of-the art SMT system available as open source has been proved to be a successful way of enabling and stimulating research in the area, as researchers do not have to invest large amounts of effort in reimplementing the work of others. In order to improve on the current best systems, they can take Moses as starting point, learn from its open code, and implement their own proposed improvements on top of it.

1.2. Motivations for New Software

As stated in the previous Section, MERT is a crucial step for optimally tuning any SMT system based on a discriminative linear model. Consequently, any possible enhancement of MERT could improve the overall performance of an SMT system.

Moreover, an effective and efficient implementation of MERT is fundamental per se, and essentially independent from the MT engine. In fact, weight optimization is still an open issue

which the MT community continues researching on, especially in regard to convergence issues. The availability of a flexible and modular open source software could help this research.

Finally, the original implementation of MERT provided with Moses is a collection of scripts written in several programming language and in different periods. The interaction of the modules is not optimal, and consequently its efficiency is quite limited. The old version of MERT strongly relies on BLEU (Papineni et al., 2001) as an automatic MT measure, and on the weight optimization criterion proposed by (Och, 2003); adding new automatic MT measures or new optimization algorithms would have been hard.

For these reasons, during the Second MT Marathon[1] held in Berlin in 2008, it was decided to implement MERT in a new standalone open-source software and to isolate it from Moses as much as possible. The new implementation was defined from scratch aiming at i) improving efficiency in terms of computation time, runtime memory consumption, storage disk space, etc.; ii) increasing modularity and flexibility to easily allow for new scoring measures and new optimization criteria; iii) parallelizing some steps of the optimization process. The core of the software was written in C++. The new MERT software is licensed under the LGPL[2].

A detail documentation how to use the software can be found online[3].

2. Design and Implementation

2.1. System Outline

Using a small (typically 500-1000) set of parallel sentences (the *tuning set*) MERT attempts to find a set of feature weights which maximize the decoder performance on this set. The full MERT algorithm consists of an outer loop and an inner loop (as illustrated in Figure 1). The outer loop runs the decoder over the source sentences in the tuning set with a given set of feature weights, generating n-best lists of translations, and then calls the inner loop to optimize the weights based on those n-best lists, repeating until the weights no longer change. In the inner loop, an iterative line optimization algorithm (Och, 2003, Koehn, forthcoming) is applied to search for the highest scoring feature weights using estimates of the decoder score derived using the n-best lists.

To ensure that the n-best lists are as diverse as possible, the n-best lists produced by each run of the decoder are merged with those produced by the previous runs. The number of previous n-best lists can be chosen at runtime.

Within Moses, the outer loop was implemented using a perl script (`mert-moses.pl`) and the original implementation of the inner loop (`score-nbest.py` and `cmert`) consisted of perl, python and C programs. The main focus of the improved MERT was the inner loop, which was completely rewritten from scratch, although a new version of the outer loop script (`mert-moses-new.pl`) was also created as it was necessary to change the interface between the inner and outer loops.

[1] `http://www.statmt.org/mtm2/`

[2] `http://www.gnu.org/licenses/lgpl.html`

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

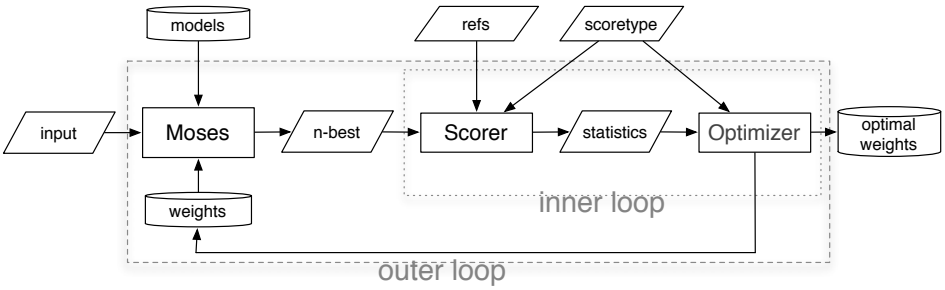

Figure 1. Outer and inner loops of MERT

Conceptually the inner loop consists of two components — the scorer and the optimizer — which are made explicit by the object-oriented design of the new MERT. The job of the scorer is to use an automatic metric (e.g. BLEU, PER, METEOR) to score a given ranking of the n-best lists, whilst the optimizer performs the actual parameter optimization. For efficiency purposes, as much of the metric calculation as possible is performed prior to running the optimization, because the optimizer will have to score a large number of translation hypothesis. For the BLEU scorer, for example, all the n-gram statistics are precalculated. The new MERT implementation is currently split into two processes; `extractor`, which does the scoring precalculations, and `mert`, which does the optimization. The `extractor` also has the job of extracting the feature values corresponding to each hypothesis in the n-best list, and making them available to the optimizer `mert`.

There is also a regression testing framework for the new MERT, which simulates the outer loop with pre-prepared data and makes it possible to check that the optimized weights agrees with the expected. The testing framework produces timing information to enable the monitoring of MERT runtime performance.

2.2. Object Model

In Figure 2 the main classes in the new MERT implementation are shown, using UML[4]. The `Scorer` class is the abstract base class of all scorers, and currently has two concrete subclasses; `BleuScorer`, which implements BLEU, and `PerScorer`, which implements position-independent recognition rate. The main work of the `Scorer` is done in the two statistics precalculation methods (`setReferenceFiles()` and `prepareStats()`) and the main scoring method (`score()`), which is used by the optimizer. In the scoring class hierarchy there is also another abstract class (`StatisticsBasedScorer`). This is used to abstract out common features of automatic translation metrics that are calculated by adding some statistics across all examples in the test set and then performing a calculation on the totals.

The optimization strategy is also encapsulated in a class, the `Optimizer`, which currently has one concrete subclass which implements the line optimization algorithm

[4]Unified Modelling Language - see `www.uml.org` for more information.

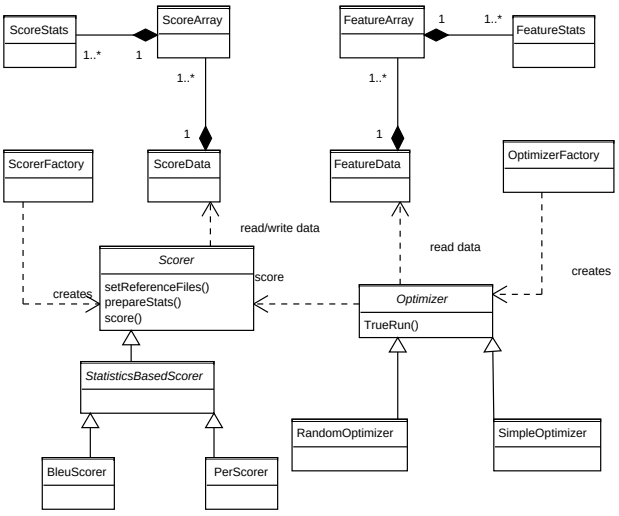

Figure 2. Class diagram for the MERT implementation

(`SimpleOptimizer`) and another which merely performs a baseline random optimization. Another optimization strategy could easily be added by subclassing `Optimizer` and overriding the `TrueRun()` method. Both the scoring strategy classes and the optimizer strategy classes have corresponding factories which are used to construct an instance of the appropriate type.

In addition to scoring and optimization, the other important set of classes are those that concern input/output, and these are the `Score*` and `Feature*` classes found at the top of Figure 2. Recall that at the start of the inner loop, the `extractor` processes the `n`-best lists, extracting scoring statistics and feature values, and passes them to the `mert` process. The file system is used to interface between these two processes, and the input/output classes are used to load and save the data in either textual or binary format. The former is easier to debug, whilst the latter offers better performance. `extractor` also transforms data from/to textual and binary formats.

3. Evaluation

In this Section we compare the old and new implementations of MERT. First, we check that they are similarly effective, i.e. that they provide similar optimized weights, and consequently achieve similar translation performance. Then, we measure the efficiency of the two versions in terms of disk occupancy and computation time. Finally, we present an add-on of MERT confirming the ease of extensibility of the new implementation.

3.1. Translation Performance

In order to verify that the new MERT works correctly, it was tested on two French-English translation tasks using data provided for the Third Workshop on Statistical Machine Transla-

tion (WMT08) (Callison-Burch et al., 2008), and the results compared to those achieved by the old MERT implementation. In the first experiment, Moses was trained on the 2007 release of the news-commentary (nc) parallel training set, using the target side of this for language modelling. The nc-dev07 set was used for tuning on 100-best lists and the decoder was tested on nc-devtest07, nc-test07 and newstest08. The BLEU scores for the old and new MERT implementations are shown in Table 1.

	nc-devtest07	nc-test07	newstest08
old mert	24.42	25.55	15.50
new mert	24.87	25.70	15.54

*Table 1. Comparison of performance (*bleu*) of old and new MERT implementations, using the news commentary training and test data.*

The second experiment also used French-English data from the WMT08 workshop, but this time Moses was trained, tuned and tested on europarl extracts. The training data was the europarl v3 release, the tuning set was dev06 and the test sets were devtest06, test06 and test07. Again both the new and old MERT implementations were used for tuning with 100-best lists. The BLEU scores are shown in Table 2. From the results of these two experiments, it can be

	devtest06	test06	test07
old mert	32.75	32.67	33.23
new mert	32.86	32.79	33.19

*Table 2. Comparison of performance (*bleu*) of old and new MERT implementations, using the europarl training and test data.*

seen that the weights learnt by the old MERT and new MERT lead to translation performances which are virtually indistinguishable.

3.2. Space and Time Performance

We also compared the efficiency of the old and new MERT implementations in terms of disk occupancy and computation time, we ran the two versions on a development data (dev06) of the Spanish-English translation tasks of WMT08 workshop. Moses was trained in a standard way on the data provided for this task. At each iteration we generated 200-best alternatives for each of the 2000 input sentences in the tuning set. Extraction of score statistics and weight optimization were performed on one 64bit Intel Xeon CPU 3.20GHz machine.

The number of n-best lists used by the new MERT software to optimize weights, is configurable at runtime: in addition to the last generated list, it can exploit from 0 to all previous ones. We ran the new MERT software in three different conditions, namely using 1, 3, and all previous n-best lists. Each iteration of the optimization process of the four MERT configurations, namely `old`, `new-all`, `new-3`, `new-1`, produces a set of weights, and hence a specific system. These systems are evaluated both on the dev06 and the test08; BLEU scores are shown in Figure 3, respectively.

For some reason to investigate more deeply, the second iteration of the new MERT produces very bad weights, which gives performance close to 0. In any case, this behavior was already observed by Macherey et al. (2008), who attribute the performance drop to an overfitting issue.

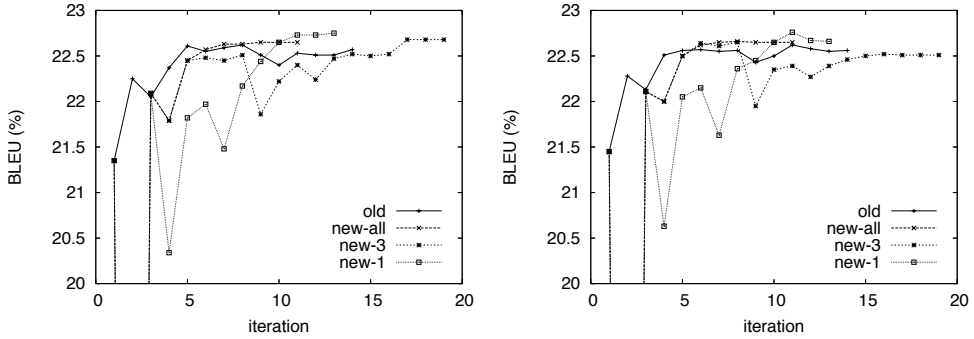

Figure 3. bleu *score of different versions of MERT on dev06 (left) and test08 (right).*

Both MERT implementations converges after 5/6 iterations, when using all the previous n-best lists; while convergence is slower and less stable if new MERT uses fewer.

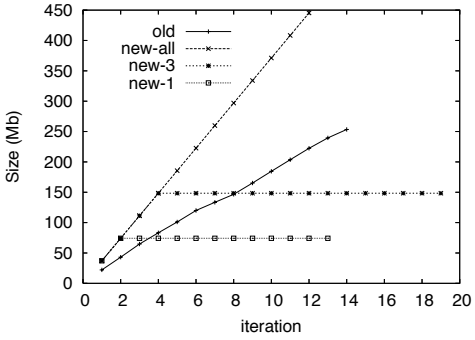

Figure 4. Global size (in Mbytes) of the files needed to the MERT implementations at each iterations.

Figure 4 reports the total size (in Mbytes) of files needed to store all the required statistics for the weight optimization at a given iteration. Figures are given for compressed[5] files for the old versions and for the binary files of the new one. New implementation requires twice a bigger disk occupancy; but limiting the number of previous n-best lists taken in consideration for optimization allows to maintain the disk usage constant.

The plot on the left of Figure 5 shows time (in seconds) to perform a single iteration, excluding time for decoding and generating the actual n-best list because independent from the

[5]Compression is performed by the gzip command of Linux with its default compression level (6).

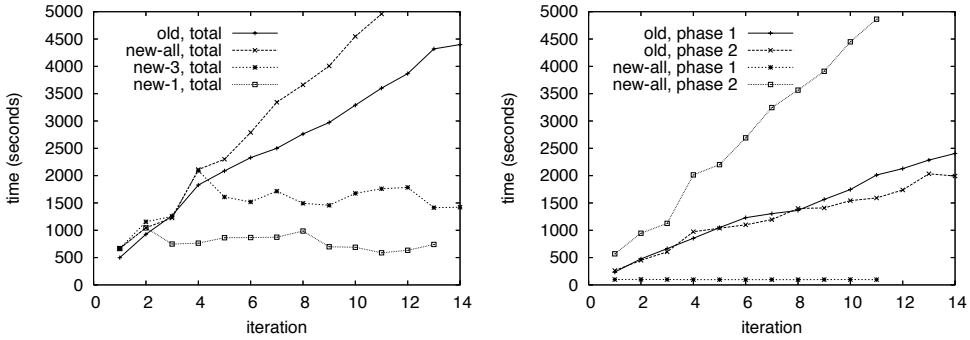

Figure 5. Time for performing a single iteration (left). Time for extracting features and computing score statistics (phase 1), and for optimizing weights (phase 2) (right).

inner loop of any MERT implementations. The plot on the right reports separately the time of the two phases which the inner loop is divided in: 1) extracting feature scores, computing score statistics and saving on the disk, and 2) optimizing weights.

In phase 1 the old MERT implementation sorts actual translation candidates and removes duplicates of those already observed in previous iterations; instead, the new one computes and stores information for all candidates of the actual n-best list. Consequently, in phase 2 the old version searches the best weights over a smaller set of candidates than the new one. Hence, the former takes a time proportional to the stored candidates in both phases; while the latter requires a constant time for the first phase and a larger time for the second one. The gap slightly expands as the number of iterations (and candidates) increases. Again, the new MERT implementation easily allows to bound computation time by taking into account fewer iterations.

It is worth noticing that new MERT software is still under development, and the removal of duplicates has the highest priority in the agenda of enhancements.

3.3. Extensibility

Since one of the aims of the re-implementation of MERT was to provide a cleaner design and so improve the extensibility of the codebase, a useful opportunity to test this extensibility was presented by some recently published improvements to the MERT algorithm. It was shown by Cer et al. (2008) that by using a form of "regularization", the risk of MERT being misled by spurious maxima (i.e. spikes in the error surface) could be reduced, leading to improvements in translation performance. The central idea is that, when performing a line search for the best BLEU score, instead of taking the BLEU scores at each point, the BLEU scores are "smoothed out" across a neighborhood of the point. This smoothing out is accomplished by one of two strategies; taking the minimum or taking the average.

When implementing the regularization method (Cer et al., 2008), it was clear that it was not just applicable to BLEU, and so was implemented higher up the class hierarchy (in `Statis-`

`ticsBasedScorer` - see Figure 2) so that it could be used with other scoring schemes. The implementation was a straightforward addition to the `score()` method.

To test the effect of the regularization on translation performance, experiments were performed using the WMT08 data for the language pairs French-English and German-English. The europarl parallel training set was used for training, with the europarl monolingual for language modelling, dev06 for tuning, and the resulting system tested on devtest06, test06 and test07. The translation scores are shown in Table 3. Unfortunately no clear pattern is visible in

		en-fr			en-de		
Method	Window	devtest06	test06	test07	devtest06	test06	test07
none	n/a	32.86	32.79	33.19	27.54	27.67	28.07
minimum	±1	32.70	32.65	33.20	27.51	27.79	28.00
	±2	32.81	32.75	33.21	27.75	27.85	28.10
	±3	32.83	32.76	32.93	27.70	27.92	27.96
	±4	32.88	32.77	33.24	27.70	27.87	28.02
average	±1	32.79	32.77	33.29	27.44	27.81	28.00
	±2	32.89	32.83	33.28	27.63	27.73	27.98
	±3	32.78	32.67	33.19	27.53	27.67	27.87
	±4	32.81	32.79	33.25	27.81	28.01	28.22

Table 3. Experiments with mert regularization

the results in Table 3. Examination of the error surfaces explored by MERT suggests that they are fairly smooth and not prone to the sort of sudden spikes that this regularisation is designed to smooth out. It is hypothesised that for unrelated language pairs (such as Chinese-English) these kind of irregularities in the error surface are more common than for related (and therefore easier) language pairs such as German-English and French-English. Further investigation would be required to confirm this hypothesis.

Developers have already added Position Independent Error Rate (PER) as an alternative automatic translation score. Results are not reported, because they essentially confirm those achieved by exploiting BLEU.

4. Conclusion and Future Work

In this paper we presented a new open-source software implementing MERT. Although it is still distributed under Moses toolkit, it is essentially a standalone piece of software. The most important characteristic of the distribution is its modularity which allows an easy extensibility to new error measures and enhanced optimization algorithms. At the moment, BLEU score and PER are implemented and two optimization criteria can be chosen: a (possibly smoothed) line-wise optimization, and a dummy random optimization.

New version of MERT actually requires more disk usage and is slightly slower than the previous one, because it does not remove duplicate translations from the n-best lists. This issue will be addressed very soon. The parallelization of portions of the algorithm is also in the close-term agenda, to reduce computational time.

In the close future, developers will also work on i) adding new automatic measures, like WER and NIST score, and combination of them, ii) constraining the space of the feature weights, iii) adding priors to weights and iv) implementing the lattice optimization as proposed by Macherey et al. (2008).

Bibliography

Callison-Burch, Chris, Philipp Koehn, Christof Monz, Josh Schroeder, and Cameron Shaw Fordyce, editors. 2008. *Proceedings of the Third Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Columbus, Ohio.

Cer, Daniel, Dan Jurafsky, and Christopher D. Manning. 2008. Regularization and search for minimum error rate training. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 26–34, Columbus, Ohio.

Koehn, Philipp. forthcoming. *Statistical Machine Translation*. Cambridge University Press.

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. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demo Session*, pages 177–180, Prague, Czech Republic.

Koehn, Philipp, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*, pages 127–133, Edmonton, Canada.

Macherey, Wolfgang, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of EMNLP*, pages 725–734, Honolulu, Hawaii.

Moore, Robert C. and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of Coling*, pages 585–592, Manchester, UK.

Och, Franz Josef. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167, Morristown, NJ, USA.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. Bleu: a method for automatic evaluation of machine translation. Research Report RC22176, IBM Research Division, Thomas J. Watson Research Center.