MOSES CORE

Deliverable D1.4

Moses Version 3.0 Release Notes

Work Package:	WP1: Moses Coordination and Integration
Author:	Hieu Hoang
Internal Reviewer:	Nicola Bertoldi
Due Date:	January 31st, 2015
Dissemination Level:	Public

May 28, 2015

Moses Version 3.0 Release Notes

Overview

The document contains the release notes for the Moses SMT toolkit, version 3.0. It describes the changes to the toolkit since version 2.0 from January 2014.

In the previous version, the internal feature function framework was updated. This framework allows arbitrary feature functions to be easily added that can score partial and completed translations during decoding. This was a major undertaking which we hope will serve the Moses community well into the future by giving researchers and developers a framework with which they can easily extended Moses.

Moses 3.0 will be the last tested release for the forseeable future. Therefore, the theme of this release is consolidation of the existing codebase to ensure that it will be stable and useable once the MosesCore project finishes. The feature function framework has proven popular with developers and has been extended according to requests. The two mainline decoder executables, moses and moses_chart, have been merged and duplicate code removed.

We have also taken over the maintenance of the mgiza project. This project implements the fast multi-thread version of GIZA++ which produces word alignment, an essential step in the Moses training pipeline. By adding mgiza as a sister project to Moses under the same administrative umbrella, we hope that it will be maintained and be useful to the community for many years to come.

New Features

The following is a list of the major new features in the Moses toolkit since version 2.0, in roughly reverse chronological order.

1. Faster decoder when using sparse features by Matthias Huck. Prior to the change, scores for all language models were copied to new hypotheses when a hypothesis is extended. This is suitable for a small number of dense scores but the copying is inefficient a large number of sparse scores. Now, only the total weighted scores are copied during hypothesis expansion and only changes in the score as stored in each hypothesis.
2. Faster, more memory efficient synchronous context-free grammar (SCFG) decoding (Sennrich, 2014) by Rico Sennrich. The decoding algorithm, CYK+ (Chappelier, 1998), requires intermediate data structures which consume a lot of memory. By changing the order in which subphrases of the input sentence

are decoded, it is possible to get rid of these data structures. This not only result in less memory consumption but increase decoding speed as less memory has to be allocated.

3. Merging moses and moses_chart by Hieu Hoang. As per the MosesCore WP1, Task 1.6, the two mainline decoder executables, moses and moses_chart, have been merged. This creates one executable for both phrase-based and SCFG models. However, we have gone beyond the deliverable by creating a framework to enable other translation models and decoding algorithms to be easily added. This will make unnecessary the divergence which caused the split between the phrase-based and SCFG models. The framework has been put to immediate use as more complex STSG (Synchronous Tree Substitution Grammar) have recently been added to Moses.
4. Synchronous Forest and Tree Substitution Grammar (STSG) models by Phil Williams. The existing syntactic translation models in Moses are based on SCFG. Following (Huang et al, 2006), STSG models have been added to Moses. These are richer syntactic models which it is hoped will lead to better, syntactically correct output.
5. Smaller phrase-table by pruning according translation model scores only by Philipp Koehn. Many low scoring translation rules are created due to misalignments and non-parallel text in the parallel corpora. These rules are very rarely used but consume a large amount of disk space and cause degradation in decoding speed. These problems are easily solved by discarding them at training time.
6. Simulated post editing tools by Ulrich Germann (German, 2014). This simulates the updating of the dynamic suffix array phrase-table that occurs when Moses is used in a computer-aided translation (CAT) tool.
7. Hypergraph batch-mira by Barry Haddow. Following (Cherry and Foster, 2012), the search graph from the decoder can be used as the input to the batch-mira tuning algorithm.
8. Dynamic Suffix-Array phrase-table by Ulrich Germann (German, 2014). The standard Moses training regime creates a phrase-table containing all translation rules that have been extracted from an aligned parallel corpus. By contrast, a suffix array phrase-table stores the aligned corpus and extracts the rules on-demand for a given input sentence. This reduces the training time and reduces memory consumption required to store the full size phrase-table. Also, the *dynamic* feature of the suffix array allows new parallel sentences to be incorporated into the corpus and which can then update the translation rules that can be extracted.
9. OxLM, a log-bilinear language model, by (Baltescu et al, 2014). Neural network feature functions have been gaining popularity in the SMT community in the past few years. The OxLM implement a neural network language model for the Moses decoder.
10. Bilingual language model (Devlin et al, 2014) by Nikolay Bogoychev, Paul

Baltescu, Rico Sennrich, and others. Building on OxLM and the other neural network language models such as by (Vaswani, 2013), feature functions have been created which scores the translation, given both source and target context.

11. Vowpal Wabbit (VW) Classifier-based feature function by Ales Tamchyna and Marcin Junczys-Dowmunt. Extending (Carpuat, 2012), feature functions were added to the decoder that scores translation rules, conditioned on other possibles rules for the same input subphrase.
12. Cache-based Translation Model and Language Model by Nicola Bertoldi (Bertoldi, 2014). These features can be dynamically modified by adding, removing, and re-scoring entries according to a time-decay function, without the need of re-starting the decoder. They are suitable for an online translation framework like the Computer Assisted Scenario CAT scenario and aim at rewarding or penalizing some options according to any external feedback. In particular, the Cache-based Translation Model works like any other phrase table providing the decoder with new translation rules.
13. Daily speed testing framework. Described in Section Speed Testing, below.

Extensions to the Feature Function Framework

The feature function framework implemented in release 2.0 has been extended, based on user feedback.

1. A method, ChangeSource(), can be overridden by any feature function in order arbitrarily change the input sentence. For example, by integrating a tagger or parser into the decoder, a feature function can add linguistic information to the input.
2. Feature functions return scores which are used by the decoder to assess the quality of the translation. This assessment is done in combination with many other feature functions, it is the decoder which decides whether a translation should be used. However, in some circumstances, a feature function can unilaterarly decide to prune a particular translation. Example of uses include feature functions which consults blacklists or terminology databases to prune inappropriate translations.
3. Feature functions typically evaluate translation rules independently of alternative rules that can translate the same source words. We have extended the feature function framework to allow feature functions to score rules, given knowledge of alternative rules. This has been used by a new feature function which is discriminatively train to pick the best rules.

End-to-End Testing and Pre-Made Models

As with version 1.0 and 2.0, a number of full-scale experiments were run. This is the final test to ensure that the Moses pipeline can run from beginning to end, uninterrupted, with 'real-world' datasets. The translation quality is noted, as measured by BLEU, to ensure that there is no significant variation in performance due to any interaction between components in the pipeline.

The end-to-end tests produce a large number of tuned models. The models, as well as all configuration and data files, are made available for download. This is useful as a template for users setting up their own experimental environment, or for those who just want the models without running the experiments.

The URL for downloading the pre-made models is:

http://www.statmt.org/moses/RELEASE-3.0/models/

To use the pre-made model, the following steps need to be taken:

1. Download the tuned moses ini file should from the directory `tuning/`
2. Download the model files referenced in the ini file.
3. Binarize the model files according to documentation on the Moses website (http://www.statmt.org/moses/)
4. Change the ini file to use the binary model files, instead of the initial text models
5. Download the corresponding recaser from the directory `recasing/`
6. Tokenize and lower case your input, translate with the decoder, then recase and detokenize your output.

Performance

The Moses regression tests were run to ensure that the output remained constant despite the major refactoring changes. 22 new regression tests were created to improve the test coverage.

End-to-end testing of the Moses pipeline was run for 9 European language pairs, trained on the Europarl corpus, to ensure that translation quality had been maintained. The tests were initiated for Moses 0.91 and have been expanded to test new features in version 2.1. There seems to be no consistent degradation in the translation quality, besides random variation. The results are shown in Appendix A.

Speed Testing

In order to make sure that commits to Moses do not adversely affect speed (such as the bug that was discovered between Release 2.1 and 2.1.1), a testing regime was implemented to monitor the speed performance of critical components in the Moses training and decoding

pipeline. The testing is automatically carried out on a daily basis, on a dedicated server to reduce the variation caused by other tasks running on the same machine.

The results of the tests are available online:

http://www.statmt.org/speed-test/

Appendix B shows the speed test results for the codebase as of the 28th January, 2015, close to Moses version 3.0. From 45 tests, 16 improved by 5% or more over Release 2.1.1, 28 showed less than 5% change either way, and only 1 slowed by more than 5%.

Cross-Platform Compatibility

As with previous releases, we ensured that Moses, IRSTLM, and MGIZA compile on a number of operating systems commonly used by the community. We test them on the following platforms:

i. Windows 8 (32-bit) with Cygwin 6.1
ii. Windows 10 (64-bit Preview edition) with Cygwin 6.1
ii. Mac OSX Yosemite with MacPorts
iii. Ubuntu 14.04 (32 and 64-bit)
v. Fedora 20 (64-bit)
vi. Fedora 21 (64-bit)

Binaries for all platforms are made available for download for users who do not wish to compile their own version. The URL for downloads is

http://www.statmt.org/moses/RELEASE-3.0/binaries/

We also make available the 32 and 64-bit Linux virtual machines with Moses, IRSTLM, and MGIZA pre-installed at the following URL:

http://www.statmt.org/moses/RELEASE-3.0/vm

There are a number of Issues on some platforms:

1. Regression tests are only guaranteed to pass on Ubuntu 12.04 (64-bit), since different compiler versions and different versions of the standard library can lead to differences in the way floating point numbers are rounded. See, for example http://www.exploringbinary.com/correctly-rounded-conversions-in-gcc-and-glibc/
2. End-to-end testing only tested on Linux (Ubuntu 12.04 64-bit).
3. Unit test for BackwardLM fails on Windows/Cygwin (32-bit and 64-bit). This is because the test as written relies on the shared boost testing libraries, which are not available on Cygwin.

MGIZA

MGIZA is the multi-threaded version of the GIZA++, the popular word alignment tool. It is able to make the most of modern multi-core servers by parallelizing the workload. In addition, it

also has unique features such as forced alignment using an existing model. It is the recommended word alignment tool for most users.

The core developer can no longer maintain the code, therefore, we have taken over its maintenance. MGIZA is now a sister project of Moses, administered by the same developers. The impact has been immediate. By enabling the user-feedback and transparency as has always been the case with Moses, we have identified and fixed several long-standing usability and performance issues, boosting word alignment speed by a factor of four.

Bibliography

Baltescu, Paul and Phil Blunsom and Hieu Hoang (2014), OxLM: A Neural Language Modelling Framework for Machine Translation, vol 102, pp81-92, The Prague Bulletin of Mathematical Linguistics

Bertoldi, Nicola (2014), Dynamic Models in Moses for Online Adaptation,,The Prague Bulletin of Mathematical Linguistics, vol. 101, 2014 , pp. 7 - 28

Carpuat, Marine, Hal Daume III, Alexander Fraser, Chris Quirk, Fabienne Braune, Ann Clifton, Ann Irvine, Jagadeesh Jagarlamudi, John Morgan, Majid Razmara, Ales Tamchyna, Katharine Henry and Rachel Rudinger (2012). Domain Adaptation in Machine Translation: Final Report. 2012 Johns Hopkins Summer Workshop Final Report

Chappelier, J. C. and M Rajman (1998). A Generalized CYK Algorithm for Parsing Stochastic CFG. *TAPD*, *98*(133-137), 5.

Cherry, Colin and George Foster (2012), Batch Tuning Strategies for Statistical Machine Translation, in Proceedings of NAACL

Devlin, J., R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul (2014). Fast and robust neural network joint models for statistical machine translation. In Proceedings of ACL

Germann, Ulrich (2014). Dynamic Phrase Tables for Machine Translation in an Interactive Post-editing Scenario, In Proceedings of the Workshop on Interactive and Adaptive Machine Translation, pages 20-31, 2014.

Huang, L., K. Knight, and A. Joshi (2006). A syntax-directed translator with extended domain of locality. In Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing, pages 1–8, New York City, New York. Association for Computational Linguistics.

Sennrich, Rico (2014). A CYK+ Variant for SCFG Decoding Without a Dot Chart In: Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8). Doha, Qatar, pp. 94-102.

Vaswani, Ashish, Yinggong Zhao, Victoria Fossum, and David Chiang, (2013). Decoding with large-scale neural language models improves translation.In Proceedings of EMNLP

Appendix A - Comparison of BLEU scores with previous versions of Moses

- Red numbers refer to the best performance
- Since online MIRA is no longer available in V3.0 (it was superseded by k-best MIRA) it was not tested.

			RELEASE 0.91	RELEASE 1.0	RELEASE-2.1	RELEASE-3.0
En-es	1	pb	24.70	24.81	24.61	24.66
	2	hiero	24.18	24.20	23.74	24.45
	3	(1) + placeholders			24.80	24.59
	4	(1) + CreateOnDiskPt			24.67	24.74
Es-en	1	pb	22.87	23.01	22.97	22.97
	2	hiero	22.32	22.37	22.20	22.26
	3	(1) + placeholders			22.78	22.65
	4	(1) + CreateOnDiskPt			23.06	22.83
En-cs	1	pb	10.98	11.04	11.16	11.03
	2	hiero	10.92	10.93	10.90	10.80
	3	(1) + placeholders			10.57	10.67
	4	(1) + CreateOnDiskPt			11.12	10.88
	5	(2) + Ken's incre algo			10.76	10.42
Cs-en	1	pb	16.00	15.72	15.81	15.76
	2	hiero	15.89	15.68	15.66	15.68
	3	(1) + placeholders			15.65	15.58
	4	(1) + CreateOnDiskPt			15.80	15.83
	5	(2) + Ken's incre algo			15.43	15.70
En-de	1	pb	11.72	11.87	11.71	11.61
	2	hiero	11.59	11.62	11.45	11.44
	3	(1) + CreateOnDiskPt			11.64	11.80
	4	(1) + POS de	11.55	11.67	11.75	11.71
	5	(4) + POS en	11.64	11.75	11.69	11.79
De-en	1	pb	15.71	15.75	15.80	15.75
	2	hiero	15.74	15.53	15.56	15.91
	3	(1) + POS de	15.71	15.84	15.70	15.91
	4	(3) + POS en	15.84	15.93	15.94	15.80
	5	(1) + CreateOnDiskPt			15.70	16.01

	6	(3) + CreateOnDiskPt			15.76	15.92
	7	(4) + CreateOnDiskPt			15.79	15.88
En-fr	1	pb truecase	24.43	24.38	24.45	24.44
	2	pb recase	22.95	22.94	22.67	22.64
	3	(2) + hiero	22.35	22.28	22.11	22.36
	4	(2) + POS en	23.34	23.05	23.06	22.93
	5	(2) + kbmira			22.81	23.00
	6	(2) + pro			22.52	22.37
	7	(5) + mira			21.59	n/a
	8	(2) + CreateOnDiskPt			22.89	n/a
	9	(2) + CompactPt			22.67	22.69
Fr-en	1	pb truecase	24.09	24.06	23.87	23.88
	2	pb recased	22.40	22.41	22.43	22.42
	3	(2) + hiero	18.19	18.05	17.78	17.94
	4	(2) + en POS	22.51	22.55	22.47	22.48
	5	(2) + kbmira			22.46	22.52
	6	(2) + pro			22.44	22.56
	7	(2) + CreateOnDiskPt			22.50	n/a
	8	(2) + CompactPt			22.58	22.50

Appendix B - Speed test results as of 28th January, 2015

The naming of the speed tests generally follows a common pattern, with several fields separated by underscores. The fields are:

- Phrase table type - can be “ondisk”, “binary” or “minpt” - the last is for the compact phrase table.
- Reordering model type - If the translation system is phrase-based, the reordering model can be “reord” (for the original binarised version) or “minreord” (for compact) . For a hierarchical (hiero) translation model the reordering type is “hierarchical”.
- Number of cores used, e.g. “4core” for 4 cores. If this is missing, assume all cores (8).
- Whether or not the experiment was run with the models already in the operating system’s disk cache. This was achieved by running the experiment twice, and timing the second run, and such tests are marked as “vanilla_cached”. If the experiment was just run once with no caching, it is marked as “vanilla”.

In addition, there are two types of test which do not fit this naming convention:

- memory_string2tree_* - Testing of a string to tree model with an in-memory phrase table.
- score.hiero_* - Testing of hiero scoring code.

(Bracketed, red numbers refer to a worsening of speed.)

Test name	Master (sec)	Release 2.1 (sec)	Change (%)
ondisk_reord_4core_vanilla_cached	37.08	36.62	(1.26)%
ondisk_reord_1core_vanilla_cached	104.58	104.71	0.12%
binary_minreord_4core_vanilla_cached	43.31	45.34	4.48%
binary_reord_4core_vanilla	44.96	45.59	1.38%
ondisk_minreord_vanilla	34.72	40.86	15.03%
ondisk_minreord_4core_vanilla_cached	35.52	36.42	2.47%
minpt_reord_vanilla_cached	17.98	17.92	(0.33)%
minpt_reord_4core_vanilla_cached	21.38	20.84	(2.59)%
ondisk_reord_4core_vanilla	37.07	36.23	(2.32)%
ondisk_hierarchical_vanilla_cached	77.1	133.72	42.34%
ondisk_hierarchical_vanilla	77.28	254.45	69.63%
ondisk_minreord_4core_vanilla	35.88	36.26	1.05%
ondisk_hierarchical_4core_vanilla	91.29	115.47	20.94%
binary_reord_vanilla	39.25	51.17	23.29%
binary_minreord_1core_vanilla_cached	110.98	113.31	2.06%
ondisk_minreord_vanilla_cached	35.05	35.08	0.09%
ondisk_hierarchical_1core_vanilla	250.36	273.77	8.55%
ondisk_minreord_1core_vanilla_cached	105	104.37	(0.60)%
binary_minreord_vanilla	35.13	48.8	28.01%

score.hiero_vanilla_cached	129.99	130.97	0.75%
binary_reord_4core_vanilla_cached	44.91	45.85	2.05%
ondisk_minreord_1core_vanilla	103.78	104.03	0.24%
minpt_reord_1core_vanilla_cached	61.6	60.42	(1.95)%
binary_reord_vanilla_cached	39.03	44.51	12.31%
minpt_minreord_1core_vanilla_cached	60.88	61.27	0.64%
memory_string2tree_vanilla	34.71	93.43	62.85%
minpt_reord_vanilla	18.19	25.77	29.41%
ondisk_hierarchical_1core_vanilla_cached	250.29	274.83	8.93%
minpt_minreord_vanilla	15.21	21.29	28.56%
minpt_reord_1core_vanilla	61.53	60.09	(2.40)%
ondisk_reord_vanilla	37.72	44.5	15.24%
minpt_minreord_vanilla_cached	15.37	17.74	13.36%
minpt_minreord_1core_vanilla	60.81	60.93	0.20%
minpt_minreord_4core_vanilla	19.82	20.39	2.80%
binary_reord_1core_vanilla_cached	111.2	113.6	2.11%
ondisk_reord_vanilla_cached	37.21	34.85	(6.77)%
score.hiero_vanilla	129.91	131.26	1.03%
ondisk_hierarchical_4core_vanilla_cached	90.99	115.54	21.25%
ondisk_reord_1core_vanilla	104.17	104.4	0.22%
minpt_reord_4core_vanilla	21.72	20.77	(4.57)%
binary_minreord_vanilla_cached	36.17	43.91	17.63%
binary_minreord_4core_vanilla	43.6	45.28	3.71%
binary_minreord_1core_vanilla	109.91	112.63	2.41%

binary_reord_1core_vanilla	111.92	113.99	1.82%
minpt_minreord_4core_vanilla_cached	20.2	20.25	0.25%